Modèle de Gestion Hiérarchique Distribuée pour la Reconfiguration et la Prise de Décision dans les Équipements de Radio Cognitive

Loig Godard

N^o d'ordre : 3850

Thèse

présentée devant

l'**Université de Rennes 1**

pour obtenir le titre de

Docteur de l'université de Rennes 1

spécialité : *Électronique*

Modèle de Gestion Hiérarchique Distribuée pour la Reconfiguration et la Prise de Décision dans les Équipements de Radio Cognitive

par

Loïg GODARD

Soutenue le 18 / 12 / 2008 devant la commission d'examen :

Rapporteurs	M. Apostolos Kountouris	Ingénieur de recherche Orange Labs
	M. Jean-Luc Dekeyser	Professeur des universités Université de Lille
Examinateurs	Mme Fabienne Uzel-Nouvel	Maître de conférence INSA
	M. Olivier Sentieys	Professeur des universités Université Rennes 1
	M. Christophe Moy	Professeur SUPELEC
	M. Jacques Palicot	Professeur SUPELEC

SUPELEC
Institut d'Électronique et de Télécommunications de Rennes
Signal, Communication et Electronique Embarquée

À tous ceux qui rêvent encore

Remerciements

En premier lieu, je tiens à remercier Christophe MOY pour m'avoir encadré durant mes trois années de thèse et éclairé de ses précieux conseils. Je tiens aussi à remercier Jacques Palicot, responsable de l'équipe SCEE, pour m'avoir accueilli et avoir pris de son temps pour participer à diverses discutions relatives à cette thèse. Ces discussions, lors desquelles chacun exprimait son point de vue et qui finissaient par prendre Christophe à partie pour trancher pour l'une ou l'autre des propositions, m'ont énormément apportées d'un point de vue personnel. Je les remercie tous deux pour m'avoir permis de prendre part à diverses activités en dehors du simple cadre de cette thèse en me faisant toute confiance sur ma possibilité d'effectuer ces activités en parallèle avec mon travail de thèse.

Je tiens également à remercier l'ensemble des membres de mon jury avoir accepté de juger et évaluer ces travaux de thèse. Je remercie sincèrement Olivier Sentieys, Professeur des universités Université Rennes 1 à l'ENSSAT de Lannion d'avoir présidé le jury lors de ma soutenance. Je remercie également Apostolos Kountouris Ingénieur de recherche Orange Labs ainsi que Luc Dekeyser Professeur des universités Université de Lille pour l'attention qu'ils ont accordés à la lecture de ce manuscrit ainsi que pour leur participation au jury. Je remercie aussi Fabienne Uzel-Nouvel Maître de conférence à l'INSA de rennes pour avoir pris de son précieux temps pour assister à ma soutenance.

En outre je remercie aussi particulièrement Didier Vojtisek ainsi que Gilles Perrouin, membres de l'équipe Triskell à l'INRIA Rennes, pour m'avoir initié à Kermeta et avoir toujours répondu présent lors de l'élaboration du simulateur HDCRAM.

Pour finir je remercie l'ensemble des membres de l'équipe SCEE (permanents, post-doctorants et doctorants) pour leur bonne humeur au quotidien et à la bonne ambiance générale au sein de l'équipe qui m'ont beaucoup apportés tant sur le plan personnel que professionnel. Ainsi, j'aimerais remercier particulièrement Yasser l'imperturbable et grand maître rush péon, Olivier le graphiste qui a égayé notre bureau de son art, Amor pour sa joie de vivre, Julien le bout en train, Fred le maître de la compilation du noyau Free BSD, Nico, Fanch, Jérôme, Mihai et Marius pour leur convivialité et leur accueil. Je tiens aussi à remercier l'ensemble du personnel SUPELEC, tant le 5050 pour leur intervention rapide sur les problèmes informatiques que les secrétaires pour leurs rapidité dans la gestion des diverses tâches administratives. Je tiens aussi à remercier le personnel de l'équipe de restauration de la Sodex'ho pour leur bonne humeur et leur sourire au quotidien

Je ne pourrais terminer ces remerciements sans y faire mention de ma famille et de mes amis qui m'ont soutenu et épaulé durant ces trois années de thèse. Un grand merci à tous.

Table des matières

Acronymes & Abréviations

La signification d'une abréviation ou d'un sigle n'est souvent indiquée que lors de sa première apparition dans le texte. Il existe dans la plupart des cas une abréviation en français et une abréviation en anglais. Dans les deux cas, les deux abréviations sont données une première fois et nous employons ensuite l'abréviation la plus usuelle, celle-ci étant le plus souvent celle en anglais.

3G	*3^{e} Génération de téléphonie mobile*
4G	*4^{e} Génération de téléphonie mobile*
ANFR	*Agence Nationale des Fréquences*
API	*Application Programming Interface*
ARCEP	*Autorité de Régulation des Communications Électroniques et des Postes*
ASIC	*Application Specific Integrated Circuit*
ASIP	*Application Specific Instruction set Processor*
ASSP	*Application-Specific Standard Product*
ATSC	*Advanced Television Systems Committee*
BTS	*Base Transceiver Station*
CAN	*Convertisseur Analogique Numérique*
CEPT	*Conférence Européenne des Postes et Télécommunications*
CIM	*Computation Independent Model*
CNA	*Convertisseur Numérique Analogique*
CORBA	*Common Object Request Broker Architecture*
CR	*Cognitive Radio*
CRM	*Cognitive Radio Management*
CRMU	*Cognitive Radio Management Unit*
CSA	*Conseil Supérieur de l'Audiovisuel*
DSA	*Dynamic Spectrum Access*
DSP	*Digital Signal Processor*
DTMB	*Digital Terrestrial Multimedia Broadcast*
DVB	*Digital Video Broadcasting*
EDGE	*Enhanced Data rates for GSM Evolution*
FCC	*Federal Communications Commission*
FI	*Fréquence intermédiaire*
FIR	*Finite Impulse Response*
FPGA	*Field Programmable Gate Array*

GALS	*Globally Asynchronous Locally Synchronous*
GIPS	*Giga Instruction per second*
GPP	*General Purpose Processor*
GPRS	*General Packet Radio Service*
GPS	*Global Positionning System*
GSM	*Global System for Mobile communications*
HDCRAM	*Hierarchical and Distributed Cognitive Radio Management*
HDREM	*Hierarchical and Distributed Reconfiguration Management*
HSDPA	*High Speed Downlink Package Access*
INTERRAP	*Integration of Reactive Behavior and Rational Planning*
IP	*Intellectuel Property*
ISDB	*Integrated Services Digital Broadcasting*
ISI	*Intersymbol Interference*
MDA	*Model Driven Architecture*
MDE	*Model Driven Engineering*
MEMS	*Micro Electro Mechanical Systems*
NOC	*Network on Chip*
OCL	*Object Constraint Language*
OS	*Operating System*
OFDM	*Orthogonal Frequency Division Multiplexing*
OMG	*Object Management Group*
P-HAL	*Physical-Hardware Abstraction Layer*
PIM	*Platform Independant Model*
PSM	*Platform Specific Model*
QAM	*Quadrature Amplitude Modulation*
QOS	*Quality of Service*
REM	*Reconfiguration Management*
REMU	*Reconfiguration Management Unit*
RC	*Radio Cognitive*
RF	*Radiofréquence*
RFI	*Request For Information*
RKRL	*Radio Knowledge Representation Language*
RL	*Radio Logicielle*
RLR	*Radio Logicielle Restreinte*
RR	*Règlement des Radiocommunications*
RTOS	*Real Time Operating System*
SSC	*Shared Spectrum Company*
SCA	*Software Communication Architecture*
SDR	*Software Defined Radio*
TNS	*Traitement Numérique du Signal*
UIT	*Union internationale des télécommunications*
UMTS	*Universal Mobile Telecommunications System*
UML	*Unified Modeling Language*

VHDL	*Very-High-Speed Integrated Circuit Hardware Description Language*
WIFI	*Wireless Fidelity ou IEEE802.11b*
WLAN	*Wireless Local Area Network*

Introduction

L'ÈRE des radiocommunications que nous connaissons actuellement a connu ses débuts en 1876 avec l'invention du téléphone fixe par le Canadien Graham Bel, suivie onze ans plus tard par la découverte des ondes radios par le physicien allemand Heinrich Hertz et l'établissement par Guglielmo Marconi des premières liaisons radio en 1895 sur un peu plus de 2 km et en 1901 à travers l'Atlantique. Les bases étaient ainsi posées pour permettre l'établissement de communications fixes dans un premier temps puis mobiles. C'est en 1985 que la Commission Européenne imposa la norme GSM. Deux années plus tard fut signé le MOU (*Mémorandum Of Understanding*) par les exploitants des réseaux de treize pays européens pour la mise en œuvre du système cellulaire numérique européen. Il faut attendre 1991 pour que la première communication entre un abonné fixe et un mobile soit réalisée, avec le succès planétaire que l'on connait depuis.

Depuis l'effervescence ne cesse de croître et a engendré de nouveaux besoins en terme d'exigence. Un exemple en termes de nouveau service est l'explosion d'internet qui a amené naturellement la convergence vers l'internet mobile. Les limites de la norme GSM sont alors atteintes. En effet le débit de 9,6 kilobits par seconde (kb/s) défini à l'origine est loin d'être satisfaisant pour couvrir les nouveaux besoins de transferts de données et constitue un frein à la diffusion de contenus multimédias. De nouvelles normes sont alors mises au point telles que le GPRS (débit de 115 kb/s), EDGE (débit de 384 kb/s), l'UMTS dans sa version dite R99, pour Release 1999, (débit allant de 384 kb/s pour tout le monde en mode mobile et jusqu'à 2 Mb/s en situation "fixe") ou encore UMTS HSDPA qui garantit les débits de 2 Mb/s initialement prévus pour l'UMTS. L'ensemble de ces valeurs est présenté dans le tableau suivant.

TAB. 2: *Norme de téléphonie mobile et débits*

Norme	GSM	GPRS	EDGE	UMTS (R99)	UMTS (HSPA)
Technologie	2G	2.5G	2.75G	3G	3.5G
Année d'utilisation	1990	2003	2005	2004	2006
Débit maximum (kb/s)	9.6	171	384	2000	3600
Débit réel (kb/s)	9.6	30	177	384	3400

La croissance du nombre d'utilisateurs (en France : de 11 millions en avril 1999 à plus de 52 millions en septembre 2007, plus de 3 milliards d'abonnés dans le monde fin 2007) ainsi que l'apparition de nouvelles normes conduisent à une situation problématique dans la gestion du spectre des radiofréquences (RF) avec un fort encombrement

et la présence d'interférences entre les systèmes. En 2002 l'organisme de régulation et de gestion du spectre aux États-Unis, la *Federal Communications Commission* (FCC), a créé un groupe de réflexion devant travailler sur la pénurie des ressources fréquentielles.

Les mesures préliminaires effectuées par ce groupe mettent en évidence le besoin de plus de flexibilité dans la gestion du spectre. En effet, alors que certaines bandes sont intensivement utilisées d'autres ne le sont que peu ou pas du tout, comme le montrent les trois figures suivantes qui ont été réalisées par la *Shared Spectrum Company* (SSC [1]) dans la ville de New York le premier septembre 2004.

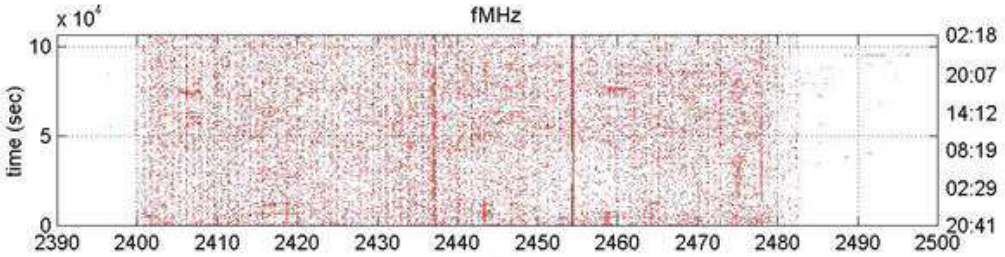


FIG. 1: *Mesures d'occupation de la bande 2390 MHz - 2500 MHz*

Il apparaît très nettement sur la figure 1 une occupation très importante des bandes de fréquences s'étendant de 2,39 à 2,5 GHz alors que la figure 2 montre une sous-occupation de la bande allant de 700 à 800 MHz aux mêmes heures. Par conséquent, il serait envisageable de mettre en œuvre des accès de type opportuniste permettant l'occupation de ces trous dans les bandes de la figure 2 par des utilisateurs des bandes de la figure 1.

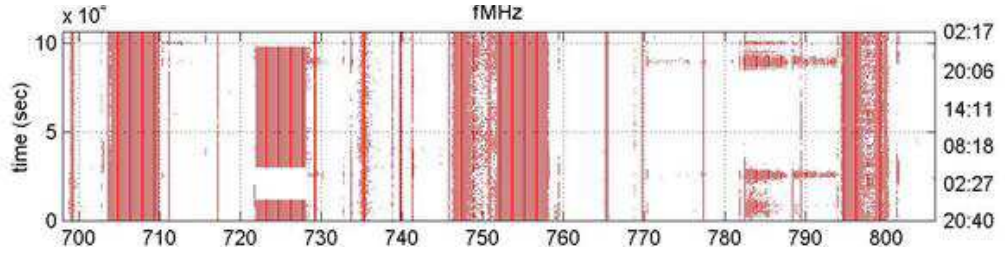


FIG. 2: *Mesures d'occupation de la bande 700 MHz - 800 MHz*

Ceci est à pondérer par le fait que les bandes non occupées ne sont pas forcément non utilisées. Comme les bandes réservées pour la radio-astronomic, les radars météo, etc. De même, est-ce qu'une bande radar peut être considérée comme inutilisée dans des endroits ou le radar ne porte pas ? Il ne faut surtout pas la perturber avec d'autres signaux. Cette nouvelle possibilité d'utilisation des ressources spectrales pourrait donc apparaître à condition que les politiques d'accès à cette ressource le permettent. L'attribution des ressources spectrales suit en effet depuis une centaine d'années une approche immuable. Ceci en raison principalement des technologies électroniques qui permettaient de mettre en œuvre les liens radio. En effet, jusqu'à présent une allocation statique des fréquences

était systématiquement utilisée : chaque application (ou service) a une bande qui lui est alloué définitivement et exclusivement. On peut par exemple citer la bande 88-108 MHz pour la radio FM ou encore 925-960 MHz pour le GSM 900. Ces bandes ne sont pas attribuées au hasard et sont règlementées à différents niveaux. L'organisation responsable au niveau mondial est le Règlement des Radiocommunications (RR) de l'UIT (Union internationale des télécommunications) qui affecte les bandes de fréquences à des services définis par grandes catégories [Transmission de données, télémesure, télécommande, Téléphonie (y compris la radiodiffusion sonore), Télévision (vidéo), etc.]. Au niveau européen, la Conférence Européenne des Postes et Télécommunications (CEPT) permet de coordonner les différents pays membres pour la normalisation des radiocommunications. En France, c'est le rôle de l'Agence Nationale des Fréquences (ANFr) d'allouer les fréquences. Cette allocation est faite en accord avec :

- les départements ministériels ayant des besoins propres (Défense, Intérieur, Aviation civile, Météo, Espace, etc.),
- le conseil supérieur de l'audiovisuel (CSA) pour la diffusion audiovisuelle,
- l'autorité de régulation des communications électroniques et des postes (ARCEP) pour les communications électroniques).

La figure 3 montre de façon évidente qu'une grande partie du spectre radio n'est pas utilisée efficacement.

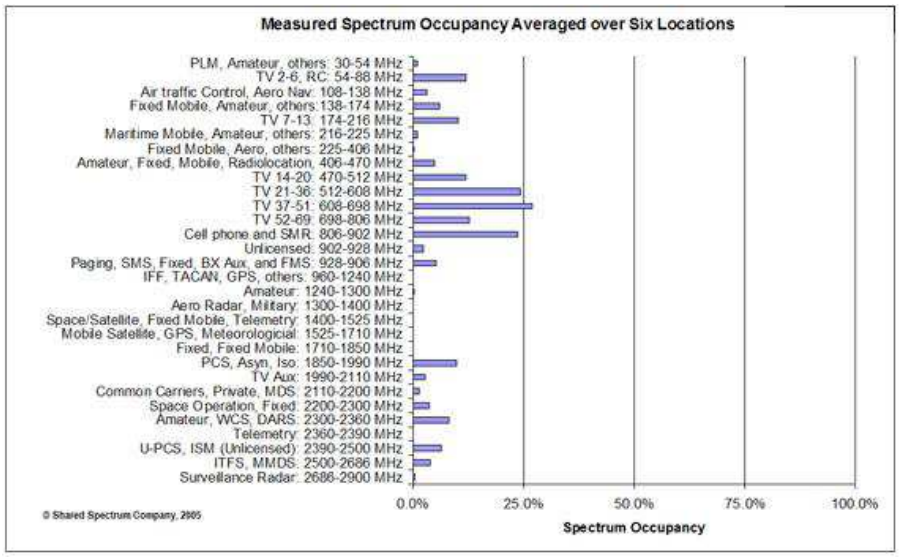


FIG. 3: *Mesures d'occupation du spectre dans six villes des ÉTATS-UNIS*

Il est donc clair que la pénurie des fréquences n'est qu'artificielle et peut être régulée par une nouvelle politique d'accès plus flexible. En effet, à un moment et à un endroit donné, en parcourant le spectre entièrement, il est possible de trouver une bande de fréquences non utilisée par son propriétaire. Par la mise en service de moyens spécifiques tels que l'accès dynamique au spectre (*Dynamic Spectrum Access* ou DSA), il serait alors

possible d'accéder à cette ressource non utilisée. On appel cela l'accès opportuniste au spectre et il permet un accès dynamique au spectre en vue d'optimiser l'utilisation. Les moyens qui permettraient sa mise en œuvre sont divers tels que la détection de bandes de fréquences libres par le terminal de communication [2], l'identification par géolocalisation [3], [4] ou encore l'identification par signal de contrôle [5].

Quelle que soit la méthode retenue afin d'exploiter ces bandes de fréquences libres, le terminal radio devra posséder une interface de communication reprogrammable afin de s'adapter aussi bien en termes de fréquence, de débit, de modulation, etc. Cette possibilité de reprogrammation est permise par une approche flexible du traitement radio apportée par la radio logicielle (RL) [6], [7]. L'approche RL a pour idéal la numérisation du traitement radio juste après l'antenne de réception réciproquement la conversion numérique/analogique avant l'antenne à l'émission. Cette numérisation permet d'une part de rassembler toutes les différentes chaines de traitements spécifiques à un standard radio sur des unités d'exécution communes au lieu de composants spécifiques à chaque fonction de chaque chaîne. D'autre part les logiciels utilisés peuvent être mis à jour ou de nouvelles versions peuvent être téléchargées afin de corriger un bogue ou de permettre l'utilisation de nouvelles fonctionnalités voire d'un nouveau standard radio. En effet, une chaîne de traitement entièrement numérisée permet d'utiliser des processeurs de traitements généraux qui peuvent effectuer une multitude de traitements différents. Ces processeurs exécutent des logiciels et peuvent donc, suivant le logiciel utilisé, effectuer un traitement pour l'un ou l'autre des standards radios. Il est possible de faire une analogie avec le monde informatique qui, avec un ensemble de composants identiques, peut exécuter des applications diverses suivant le logiciel qu'ils exécutent (traitement de texte, lecteur video, jeu, etc.). Cependant, afin de permettre l'utilisation d'une telle technologie dans un équipement radio il faut mettre en place une architecture permettant la gestion de reconfiguration de l'équipement de manière efficace : c'est un point central d'étude de la RL.

Suivant le type de traitement radio à effectuer il est nécessaire d'utiliser différents composants matériels. En effet technologiquement, nous sommes encore loin de pouvoir faire effectuer tous les traitements nécessaires par un seul composant généraliste de type processeur comme ceux proposés par Intel, AMD ou encore IBM. Ceci est encore plus critique dans un contexte embarqué sous fortes contraintes de consommation. Ainsi, certains traitements nécessitent plus de ressources de calcul que d'autres ou encore plus de mémoire. Des composants de traitement différents répondent à cette variété de contraintes. Une plateforme regroupant des unités de traitement de différentes natures (GPP, DSP, FPGA, etc.) est dite hétérogène. Si l'on fait abstraction des difficultés techniques de réalisation et de programmation d'une telle plateforme s'ouvre une nouvelle perspective pour les équipements radio. En effet, un équipement radio logiciel peut donc se reconfigurer à volonté afin de supporter les différents standards existants aussi bien que ceux à venir. La question qui en découle ensuite est : "serait-il possible de permettre à un tel équipement de décider de lui-même de ses paramètres opérationnels afin de s'adapter aux variations de son environnement en temps réel ?" Cette question est traitée

par l'approche nommée Radio Cognitive[1] (RC ou CR en anglais pour *Cognitive Radio*). Une telle faculté donnerait notamment à l'équipement radio la possibilité de trouver des solutions aux nouveaux problèmes de communication posés par les nouvelles contraintes d'utilisation du spectre radio. La FCC a en effet édité de nouvelles règles d'utilisation du spectre dans ses "rulemaking" [8]. Ces règles proposaient pour la première fois de modifier la règle de l'utilisation d'une bande de fréquence à usage unique. Cette nouvelle perspective d'utilisation des ressources spectrales offre donc la possibilité à un équipement de découvrir une bande non utilisée à un moment et un endroit donné pour l'occuper sur une période restreinte, on parle alors d'utilisateur secondaire. Cette agilité est en partie permise par la radio logicielle qui s'est penchée sur les problèmes de reconfiguration ainsi que sur le remplacement de circuits dédiés par des circuits généralistes contrôlés par du logiciel. La RL peut donc servir de base technologique à la radio cognitive qui doit permettre l'adaptation d'un équipement mobile à l'environnement suivant l'application, de façon transparente pour l'utilisateur. En effet, l'adaptation est permise par un contrôle total des ressources matérielles ainsi qu'une architecture permettant de modifier tout ou partie des ressources de traitement en fonction des besoins.

Ainsi, nous pensons que la meilleure piste d'étude pour répondre à ces enjeux est d'utiliser une architecture basée sur l'approche RL et de lui ajouter les fonctionnalités lui permettant d'une part, de prendre conscience de son environnement et d'autre part de prendre des décisions de changement de fonctionnement suivant l'évolution de son environnement. Nous avons donc défini une architecture de gestion cognitive basée sur une architecture de gestion RL que nous présentons dans ce mémoire. Cette architecture présente une séparation des flots de données, des flots de contrôle ainsi que des flots cognitifs. Cette séparation permet de limiter le risque d'apparition de goulots d'étranglement, en termes d'échanges de données, au sein de la structure et représente une approche totalement originale.

Les structures permettant la mise en œuvre de l'approche RC sont actuellement à l'étude dans de nombreux laboratoires et génèrent un réel investissement de la part des partenaires industriels en vue des retombées économiques espérées. Chacun met en avant sa spécificité pour la résolution d'un type de problème au sein de cette approche. En effet, la RC présente un vaste champ de recherche étendu à plusieurs disciplines scientifiques. On nommera par exemple les sciences inspirées du monde biologique, les science informatiques, les science cognitives ou encore de traitement du signal. Actuellement, deux types de démarches sont à l'étude pour la répartition de l'intelligence. L'une est orientée réseaux (*network centric*) et l'autre terminal (*terminal centric*). Il est clair que les technologies actuelles ne permettent pas l'utilisation de ressources de traitement trop importantes dans un équipement de type embarqué (gestion de l'énergie, taille/poids de l'équipement, puissance de calcul, etc.) et que l'on peut donc privilégier l'implantation de l'intelligence dans le réseau. Néanmoins, nous pensons qu'à son échelle, un équipe-

[1] le terme généralement utilisé dans la littérature française est radio intelligente. Néanmoins, pour la simple raison que l'on parle de science cognitive, je ferai référence à cette approche par le terme Radio Cognitive

ment radio peut bénéficier de l'implantation d'une intelligence (même restreinte) afin de contrôler ses propres ressources. Cela devient d'ailleurs de plus en plus nécessaire au fur et à mesure que la mobilité augmente, ce qui va dans le sens de l'évolution actuelle.

Nous avons donc conçu une architecture de gestion permettant d'introduire dans la conception d'un équipement RC les moyens :

- de capture ainsi que d'interprétation de métriques,
- de prise de décision de reconfiguration suivant l'évolution des métriques,
- de gestion de la reconfiguration de l'équipement RC.

Ces moyens eux-même ne font pas partie de cette étude mais concernent d'autres thèses au sein de l'équipe. Reste à résoudre la question du choix du langage ou de l'approche de conception/modélisation. Une des contraintes principales dans le monde de la RC est la diversité des activités de recherche. Il est donc difficile de fusionner les avancées d'un domaine avec un ou plusieurs autres car à chaque domaine correspondent un langage et une modélisation spécifique. Nous avons donc décidé de modéliser notre architecture dans le langage de modélisation offrant les perspectives les plus prometteuses dans le monde de l'ingénierie et de haut niveau d'abstraction : UML (*Unified Modeling Language*). Ce choix de modélisation a été pris afin de permettre la compréhension et la réutilisation par l'ensemble de la communauté de notre architecture. La modélisation orientée objet (dont UML est la formalisation), issue des sciences informatiques, permet de représenter les principes de fonctionnement d'une architecture par la définition des relations entre objets la constituant. De plus, l'approche orientée objet (voire orientée composant) en permettant la modélisation de briques réutilisables avec leurs interfaces est particulièrement adaptée à la modélisation d'équipements RC dont les composants sont reprogrammables. Néanmoins, bien qu'apportant une plus value très importante à la conception, la modélisation UML n'étant pas simulable, il n'est pas possible d'y valider la structure opérationnelle des objets utilisés dans notre architecture RC. Nous avons alors eu recours à l'utilisation d'un métalangage de programmation permettant de simuler le comportement de notre architecture, tout en conservant un formalisme UML. Ceci est une autre originalité très marquée de ce travail de thèse.

Problématique de l'étude

La problématique de l'étude présentée dans ce document est la suivante :

"les technologies étant devenues matures pour supporter la RL, est-il possible d'offrir une **architecture de gestion cognitive** d'un équipement (téléphone, PDA, station de base etc.) permettant de prendre en compte les évolutions de l'environnement, de les intégrer dans un processus d'adaptation amenant à une reconfiguration de tout ou partie de l'équipement afin de satisfaire aux besoins utilisateurs de façon transparente ?"

Il existe très peu ou pas d'études actuellement sur ce type d'architecture de gestion avec ce niveau d'ambition, permettant de tirer partie des spécificités des plateformes hétérogènes d'exécution. L'architecture retenue devra présenter une réactivité élevée aux changements de contextes, une possibilité de reconfiguration étendue, une connaissance et un contrôle total de ses ressources de traitement ainsi qu'une interface radio générique permettant l'échange (demande, envoi) d'informations avec le réseau ou d'autres équipements à proximité.

Plan du mémoire

Nous présenterons dans le premier chapitre de ce mémoire une architecture répondant aux besoins de la radio logicielle après avoir présenté le concept de Radio Logicielle ainsi que quelques architectures de l'état de l'art.

- La naissance du concept de Radio Logicielle
- Plateformes matérielles et architectures logicielles pour la radio logicielle
- Une architecture logicielle de gestion de reconfiguration : le HDReM

Après avoir présenté la partie radio logicielle nous allons nous intéresser au cœur de ce travail de thèse qui est la création d'une architecture de gestion pour un équipement RC. l'un des premiers objectifs était de définir les besoins d'une architecture RC suivant les réalités du domaine. Nous y avons répondu après une étude sur le concept RC ainsi que sur les besoins qu'il entraîne, par une architecture que nous avons nommé HDCRAM (Hierarchical and Distributed Cognitive Radio Management). Ainis, les trois chapitres suivant vont permettre au lecteur de suivre l'évolution de notre démarche qui est illustrée par la figure 4.

Le deuxième chapitre identifie les étapes ayant mené à la réalisation de HDCRAM avec :

- La naissance du concept Radio Cognitive
- Les besoins d'un équipement radio cognitif
- Un gestionnaire d'architecture radio cognitive : le HDCRAM

Dans un troisième temps, nous avons modélisé cette architecture dans le but de s'abstraire des contraintes matérielles et de capitaliser notre savoir faire comme illustré dans la deuxième ligne de la figure 4. Pour ce faire, nous avons utilisé un langage de modélisation de haut niveau. L'évolution de cette modélisation est de permettre une simulation fonctionnelle en s'assurant que le système simulé est le même que celui que nous avons spécifié. Pour ce faire, nous avons eu recours à un métalangage de programmation. Cette étape s'est conclue par la création d'un simulateur HDCRAM comme illustré dans la dernière ligne de la figure 4. Ce simulateur est présenté dans le troisième chapitre avec :

Objectives	Means	Results
Description of : •Functionalities •Reality	CR equipment specifications State of the art	HDCRAM
Symbolic representation of the reality	Language (UML)	HDCRAM Metamodel
Verification/Refinement of the functionalities	Design specific language (Kermeta)	HDCRAM Simulator

FIG. 4: *De la conceptualisation de HDCRAM à sa définition par un métalangage de programmation*

- Une nouvelle approche de conception
- Le langage UML
- Le simulateur HDCRAM

Le simulateur HDCRAM nous a permis ensuite d'effectuer une exploration comportementale d'HDCRAM en fonction de divers scénarios RC dont certains seront présentés ici dans le quatrième chapitre :

- Présentation de l'environnement de travail
- Scénario1 : Reconfiguration d'un chemin de données
- Scénario2 : Adaptation aux variations de niveau de batterie
- Scénario3 : accès opportuniste au spectre

Nous montrons grâce à ces scénarios qu'HDCRAM répond bien à tous les besoins de gestion (remontée de métrique, prise de décision, répercussion de la reconfiguration) en offrant tous les services nécessaires au bon fonctionnement d'un équipement RC. La conclusion propose une synthèse des travaux présentés pour amener à une discussion sur les perspectives. Les annexes se composent d'une présentation sommaire d'un métalangage de programmation exécutable nommé Kermeta ainsi que du délivrable complet présentant la synthèse des travaux effectués par le groupe de travail du projet D sur la radio flexible dans le cadre du réseau d'excellence NEWCOM.

CHAPITRE

1 La radio logicielle

Sommaire

1.1 La naissance du concept de Radio Logicielle

1.1.1 Définition et généralités

En approche classique, un équipement radio ne peut supporter qu'un faible nombre de standards car cela implique la juxtaposition d'un grand nombre de circuits dédiés, plusieurs étant nécessaire pour chaque standard. De plus leur architecture est figée à la conception. L'idée de radio logicielle a pour idéal la numérisation du signal juste après l'antenne, ce qui permet le traitement des opérations (sélection du canal, filtrage, décimation, etc.) en logiciel. Il serait ainsi possible d'utiliser n'importe quel standard de communication par simple chargement de logiciel. Née dans le milieu de la recherche militaire à la fin des années 70, la RL est introduite dans la communauté scientifique pour des applications de télécommunications civiles en 1991 par J. Mitola. C'est en 1995

qu'un *Request For Information* (ou RFI) sur la radio logicielle appliquée aux réseaux de téléphonie mobile sera lancé par Bellsouth et présentera le signe du démarrage d'une activité accrue dans le domaine. L'année suivante verra le jour de ce qui deviendra plus tard (en 1999) le *Software Defined Radio Forum* [9] (ou SDR Forum) réunissant une centaine de contributeurs internationaux dont le but est de promouvoir le développement, le déploiement et l'utilisation de la radio logicielle pour les systèmes sans fils.

Comme le montre la figure 1.1, l'approche classique [10], ici dans le cas d'un récepteur, utilise un ensemble de composants spécialisés à un type de traitement. Tout ceci est à dupliquer pour chaque standard que doit supporter un équipement.

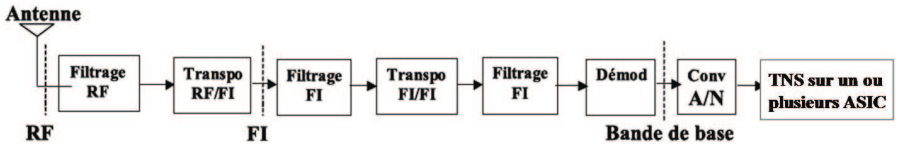


FIG. 1.1: *Architecture superhétérodyne d'un récepteur pour un standard*

Afin d'obtenir le maximum de performance pour le minimum de consommation électrique, un concepteur utilise des circuits conçus sur mesure pour un type d'application ciblé. Ce genre de composant est optimisé pour un unique type de traitement et ne peut évoluer après la conception afin de prendre en compte de nouveaux cas d'utilisation. Actuellement, au sein des équipements radio, les chaînes de transmission permettant l'accès aux standards existants sont conçues à base de tels composants nommés Circuit Intégré Spécialisé (ou ASIC pour*Application Specific Integrated Circuit* en anglais). Cette approche pour la conception des équipements radio multi-standards, utilise donc un ensemble de composants dédiés à un unique standard. Établir une liaison à l'aide de l'un ou l'autre des standards revient à activer les ASIC de la chaîne correspondant à ce standard. Un changement de standard revient à une commutation de chaîne d'ASIC. Avec ce type de méthode nommée "méthode Velcro", il est nécessaire de réaliser physiquement une chaîne de transmission différente pour chaque standard que l'on veut rendre accessible par l'équipement. Actuellement, en plus des réseaux déjà présentés auparavant dans le tableau 2, un équipement mobile doit pouvoir se connecter à des services de localisation (type GPS voir galileo un jour), des réseaux sans fils tels que le bluetooth ou le wi-fi (bientôt le wi-max), des moyens de réception TV numérique (DVB pour les européens, ATSC pour les américains, ISDB pour les japonais ou encore DTMB pour les chinois). Si l'on ajoute à cette prolifération du nombre de standards de communication l'incompatibilité des réseaux sans fils entre les différents pays on comprend aisément les problèmes de conception des nouveaux équipements radio. En effet, non seulement les nouveaux équipements se doivent de supporter les nouveaux standards ainsi que les nouvelles fonctionnalités dues à la convergence des médias (internet, musique, vidéo ...) mais ils se doivent aussi d'offrir la rétrocompatibilité avec les anciens standards. En effet, prenons le cas de la norme 3G, l'ensemble du territoire n'est pas encore couvert. Il serait donc inimaginable de demander à l'utilisateur de posséder plusieurs téléphones portables afin

d'utiliser celui qui convient à sa localisation. Le problème est que l'intégration des anciens standards a un coût en terme de surface de silicium et que l'intégration de nouveaux standards dans un équipement radio amène donc à une hausse de ce coût. Or, un équipement radio possède une taille finie et le marché révèle une forte attirance des utilisateurs pour les téléphones de plus en plus compacts. Et, bien que la technologie permette une intégration de plus en plus fine des composants (la gravure en 32 nm pour les processeurs Intel par exemple est attendue pour 2009), l'ensemble des standards existants, ajoutés aux incompatibilités de ces équipements dans différents pays, font qu'il n'est plus envisageable de concevoir de manière classique un équipement regroupant toutes ces normes. De plus, ce type de conception n'est pas évolutif et n'autorise pas la prise en compte de nouveaux standards. Ceci relève un autre problème qui est la nécessité pour l'utilisateur de changer d'équipement à chaque saut technologique afin de bénéficier des dernières avancées. Ce problème touche aussi les fournisseurs qui doivent mettre à jour les équipements au sein des stations de base afin d'offrir ces nouveaux services. Á plus long terme, on peut aussi imaginer que les équipements radio effectueront en permanence ce que l'on appel un *handover vertical*, c'est à dire passer d'un standard de communication à un autre sans interruption dans la communication, afin d'améliorer leur QoS ou la charge du réseau. On comprend alors l'intérêt pour toutes les parties (utilisateur et fournisseur) de l'intégration au sein d'un équipement de radiocommunication d'une unique chaîne de traitement, entièrement numérique comme illustré par la figure 1.2.

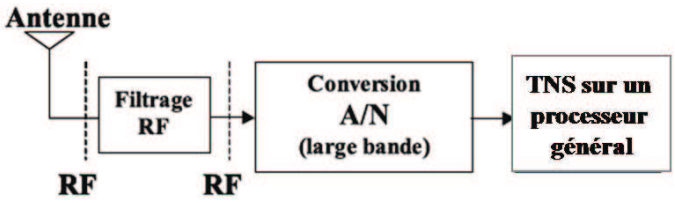


FIG. 1.2: *Architecture de radio logicielle idéale*

Cette évolution permet de se libérer de la contrainte actuelle qui est : à chaque standard un circuit dédié. On dispose alors effectivement d'un unique circuit à usage général contrôlé par du logiciel qui permet de réaliser, par traitement numérique du signal (TNS), tous les traitements de tous les standards. En effet, un équipement basé sur cette approche peut être mis à jour par modification ou ajout de nouveaux logiciels et voir ses paramètres opérationnels reconfigurés. Tout comme l'on change le fonctionnement d'un ordinateur en chargeant un nouveau logiciel ou que l'on améliore ses performances par mise à jour de logiciels. On peut aussi envisager de mettre en commun les moyens de traitement de toutes les couches protocolaires, de la couche applicative à la couche physique. Ceci permet d'envisager des optimisations inter-couches de manière beaucoup plus simple et intégrée. Néanmoins ceci est une vision plutôt ultime du concept. Prenons l'exemple d'un utilisateur émettant un appel de type visioconférence dans une zone couverte par la 3G. Si pour une raison quelconque il devait sortir de la couverture il faudrait que l'équipement puisse se reconfigure dynamiquement et sans pertes de données afin de basculer la communication en mode audio simple. Effectuer ce type de recon-

figuration en temps réel sans aucune perte semble encore difficile actuellement. Ainsi nous pensons que l'équipement devra offrir la possibilité de créer suivant les besoins une deuxième chaîne de traitement et basculer la communication afin de permettre le handover vertical. Ce traitement se rapproche du traitement classique avec le basculement physique entre deux standards hormis le fait que l'on peut se contenter d'environ deux fois la puissance de calcul nécessaire pour un standard (cas du standard le plus exigeant) afin d'offrir l'accès à un ensemble de standards pouvant être élevé, comme indiqué précédemment. En outre, l'espace utilisé par la chaîne de traitement non utilisée peut être libérée pour effectuer un autre traitement de type ponctuel. De plus cette radio logicielle idéale reste inaccessible dans la plupart des cas au vu des technologies actuelles [11]. En effet, afin de rendre ce concept réalisable il faudrait bénéficier de convertisseurs analogique/numérique disposant d'une dynamique (écart entre la puissance minimale et la puissance maximale) d'entrée trop importante pour les technologies actuelles. De plus, comme le montre [12] la puissance de traitement numérique nécessaire pour effectuer les traitements en bande de base, ainsi que la consommation sont un réel verrou technologique pour les circuits généralistes actuels. En effet, cette étude estime des performances nécessaires de 10 GIPS[1] pour le GSM jusqu'à 100 GIPS pour l'UMTS. Bien que les dernières générations de processeurs de type Intel Core 2 Extreme QX9775 soient capables de fournir une puissance d'environ 131 GIPS il n'est pas envisageable de les voir utilisés à court terme dans les stations de bases (de part leur consommation ou encore leur dissipation thermique) et encore moins dans les équipements mobiles (consommation, encombrement ou encore dissipation thermique).

Ces verrous technologiques peuvent être levés par l'adoption d'une architecture dite pragmatique [12]. Ainsi, avant une évolution future des technologies vers cette radio logicielle idéale, une étape doit être franchie avec l'adoption d'une radio logicielle dite restreinte (RLR ou SDR en anglais pour *Software Defined Radio*). Un exemple d'architecture RLR est présenté par la figure 1.3.

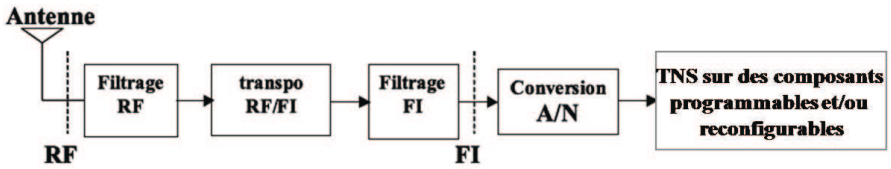


FIG. 1.3: *Architecture de radio logicielle restreinte*

Une telle structure présente différents segments technologiques au sein d'une même architecture matérielle. On trouve ainsi une partie analogique (que l'on tend à remplacer progressivement) et une partie numérique que l'on veut rendre de plus en plus importante. La numérisation se rapproche donc de plus en plus de l'antenne afin de converger vers la RL idéale. Ainsi, les circuits spécialisés (figés à la conception) sont remplacés par des circuits programmables permettant l'exécution d'applications logicielles. On peut

[1] Giga Instruction per second

noter aussi que de gros efforts sont menés également pour rendre la partie analogique conservée de plus en plus souple [13], mais ces recherches, dépassant le cadre de cette thèse, elles ne seront pas abordées ici.

1.1.2 Contraintes et besoins pour les architectures Radio Logicielle

Les contraintes liées aux architectures RLR sont nombreuses et nécessitent encore des efforts de recherche conséquents. L'une des plus importantes concerne la nature de la plateforme matérielle d'exécution cible de l'application radio conçue en logiciel ou tout du moins en numérique. En effet, une telle structure se devant d'être reconfigurable, elle se doit de disposer de ressources flexibles associant les contraintes liées aux domaines des radiocommunications ainsi que celles du domaine de l'embarqué (dans une moindre mesure pour les stations de base). Dans une approche de conception radio logicielle, une chaîne de transmission radio n'est plus constituée de composants exclusivement dédiés à un type de traitement. Il s'agit d'une suite de traitements informatiques exécutés sur des calculateurs pour faire des opérations de traitement du signal. La complexité des calculs à effectuer oblige à utiliser différents types de calculateurs mais l'approche radio logicielle est plus qu'une simple abstraction du matériel sous-jacent à l'application. C'est une approche de développement d'applications d'une façon fiable et modulaire qui permet le maximum de réutilisation des parties logicielles et matérielles d'implantation en implantation tout en laissant la possibilité de modifier les paramètres opérationnels. Les plateformes les plus adaptées à ce type de traitement sont avantageusement hétérogènes car constituées de divers éléments ayant des capacités et présentant des avantages complémentaires :

- les processeurs généraux (*GPP*),
- les processeurs de traitement numérique (*DSP*),
- les circuits spécialisés (*ASIC, ASIP, Hardware accelerator*),
- les circuits intégrés programmables (*FPGA*).

De plus une plateforme basée sur la RLR se doit de supporter le matériel de traitement RF et FI nécessaire pour effectuer le traitement du signal radio avec les ressources de calcul. Cela revient à définir une structure modifiable possédant une interface standardisée. La conception de telles plateformes est loin d'être triviale. Actuellement, le concepteur doit recourir à de nombreux logiciels et langages de programmation spécialisés lui permettant de programmer les différents composants. Il n'existe pas non plus d'environnement de travail unique permettant de concevoir de tels systèmes. Il faut aussi avoir conscience des problèmes inhérents à ce type de plateformes hétérogènes comme les différences entre chaque composant telles que :

- les vitesses de traitement différentes,
- l'équilibrage des charges de traitement,

- la gestion des ressources disponible,
- ou encore l'acheminement des données dans l'architecture.

En outre, le marché de la téléphonie mobile étant extrêmement porteur, il est normal de trouver un nombre non négligeable de constructeurs d'équipement présentant chacun un grand nombre d'équipements visant différents marchés. Dans un tel contexte, il est donc essentiel de concevoir des systèmes modulaires pouvant s'adapter à différentes plateformes et possédant des interfaces de communication standardisées. On voit donc très clairement l'intérêt de se pencher sur les sciences informatiques qui sont depuis longtemps confrontées à ce type de problème et de requérir l'utilisation de langages d'abstraction permettant de masquer les différences matérielles.

1.1.3 Quelques exemples de travaux industriels et académiques

L'hétérogénéité présentée précédemment est l'une des première préoccupations pour un concepteur d'équipement RLR : l'abstraction des contraintes d'implantation matérielle vis-à-vis de l'application radio (qui sera logicielle). Cette abstraction peut être créée par une couche intermédiaire entre le matériel et le logiciel. Ainsi, les investissements de l'armée américaine, principal moteur de la recherche dans ce domaine, a permis la définition du SCA. Le SCA, véritable référence internationale qui sera présentée plus en détail dans 1.2.4, propose un environnement facilitant le déploiement d'une architecture radio sur une plateforme matérielle quelle qu'elle soit. D'autres solutions existent comme notamment le P-HAL [14] proposé par l'Université Polytechnique de Catalogne (UPC) en Espagne. Le PHAL, qui sera présenté dans 1.3.3, propose une approche plus générale que le SCA pour la conception d'équipements RLR. Il permet non seulement une abstraction entre le matériel et le logiciel mais est aussi un flot de conception outillé. Ces outils sont en cours de conception [15].

Au niveau industriel, suite à ses travaux effectués au MIT, Vanu Bose, à créé la société Vanu Inc[2] en 1998 qui est aujourd'hui considérée comme un modèle de start up aux USA. Cette société développe et commercialise des stations de base à base de processeurs généraux de type Pentium [16]. Beaucoup de sociétés sont apparues afin de proposer des services aux concepteurs pour utiliser l'architecture logicielle SCA. On peut notamment citer le CRC (Communication Research Center of Canada) dont SCARI-Open [17] et la suite de logiciels SCARI, ont été les tous premiers environnements respectant les directives du SCA. D'autres sociétés fournissent soit des composants logiciels de l'architecture SCA ou soit des outils permettant de développer des éléments radio compatible avec le SCA, tels que Zeligsoft [18] ou encore PrismTech [19].

À la pointe de la recherche universitaire, on peut citer les laboratoires tels que Virginia Tech aux USA avec par exemple les travaux du professeur Jeffrey Reed [20], [21] ou

[2]http ://www.vanu.com/

encore les professeurs Ryuji Kohno [22] de la Yokohama National University et Hiroshi Harada [23] de la National Institute of Information and Communications Technology au Japon. En europe citons également l'Université de Karlsruhe avec le professeur F. Jondral et le Trinity Collège de Dublin avec le professeur L. Doyle. En France, Eurecom développe des plateformes RL (PLATON), le CEA conçoit des SoC et NoC pour la RL (FAUST et MAGALI). Notre équipe SCEE de SUPELEC et de l'IETR est aussi reconnue au niveau internationnal sur la thématique RL.

Au niveau collaboratif, de nombreux travaux européens ont traités des cas liés à la CR on peut nommer TRUST [24], SUNBEAM [25] le principal d'entre eux étant le projet End-to-End Reconfigurability (E2R) [5] qui a regroupé tous les principaux acteurs du domaine.

1.2 Plateformes matérielles et architectures logicielles pour la radio logicielle

1.2.1 Les ressources matérielles

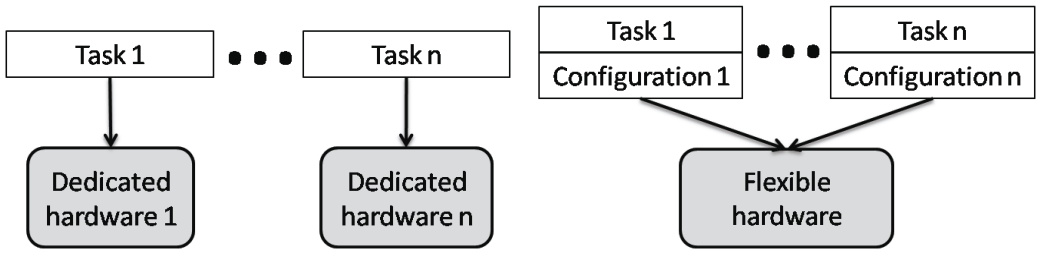
Lorsque l'on parle de plateformes dédiées à la RLR, la première idée associée est la flexibilité des ressources matérielles présentes. Cette flexibilité est la capacité de changement de contexte (changement de service, changement de traitement) offerte à un équipement et introduit les notions de reconfigurabilité, d'adaptabilité ou encore de modularité. À l'origine, les traitements radio étaient effectués sur des ASIC, ces derniers présentent de hautes performances de traitement mais leur manque de flexibilité est un lourd handicap dans le cadre d'une utilisation en RLR. À l'opposé, les circuits de type processeurs généralistes sont extrêmement flexibles. Néanmoins cette flexibilité présente un surcoût non négligeable comme des performances de calcul brutes moins élevées et une consommation énergétique plus importante. Il existe deux types de flexibilité dont nous allons parler maintenant :

- la flexibilité matérielle
- la flexibilité logicielle

Flexibilité matérielle : les circuits logiques programmables

Dans le domaine embarqué (mais aussi en électronique en général), les concepteurs sont limités par la taille maximale de silicium disponible pour la réalisation des fonctions logiques au sein d'un équipement. Dans le cadre d'une utilisation spécifique et non flexible, les ressources sont dimensionnées dans un ASIC de façon très précise afin de sa-

tisfaire les contraintes d'exécution. À l'opposé, dans une utilisation plus généraliste, les ressources sont spécifiées au pire cas et sont donc régulièrement (suivant l'ordonnancement des tâches et les possibilités de parallélisme) sous-utilisées. Les problèmes majeurs sont la consommation de ces ressources non utilisées en terme de puissance énergétique et d'occupation de surface de silicium. La consommation énergétique peut être limitée en appliquant des techniques qui permettent une gestion fine de l'horloge afin d'activer ou non certaines parties de l'architecture matérielle ou encore en mettant en place des techniques de basse consommation [26]. L'apparition de nouvelles technologies de circuits logiques programmables comme les FPGA et leurs dernières évolutions, permet de juguler ces problèmes de contraintes de surface. Comme présenté par la figure 1.4 ces circuits logiques programmables permettent de reconfigurer les ressources matérielles en fonction du type de traitement à effectuer. Cette approche offrant la flexibilité matérielle permet donc de dimensionner les ressources suivant les besoins et ce de manière dynamique (en temps réel) ou de manière statique.



(a) Dedicated hardware (b) Flexible hardware

FIG. 1.4: *Flexibilité matérielle apportée par les nouvelles technologies*

Cette reconfigurabilité offre une évolutivité importante du circuit par la possibilité d'ajouter de nouvelles fonctionnalités au cours du temps, ce qui permet une grande flexibilité d'utilisation. En effet, jusqu'à présent la flexibilité matérielle était utilisée en phase de développement comme souplesse à la conception et, par exemple, permettait à moindre coût de tester différentes solutions. Aujourd'hui on commence à étendre cette utilisation en cours de fonctionnement ou à l'allumage. Il existe deux types de reconfigurations : reconfiguration hors ligne ou *off-line* et reconfiguration en fonctionnement *on-the-fly*. La première demande une interruption du service en cours afin de procéder à la reconfiguration et nécessite un *reset* matériel avant une remise en fonctionnement, elle peut aussi être utilisée lors d'une phase d'initialisation. Le reset est appliqué afin de de s'assurer du contexte de départ de l'application (état des mémoires, des données stockées dans les registres, des données en entrée et en sortie, etc.). La reconfiguration off-line requérant une interruption de service, la notion de temps de reconfiguration n'est pas critique. Le deuxième type de reconfiguration s'effectue sans interruption du service en cours d'utilisation et peut donc être fortement soumise aux contraintes du temps réel suivant les cas d'application. Afin de permettre une reconfiguration sans interruption de

service, il est nécessaire d'utiliser une reconfiguration dite dynamique. La reconfiguration dynamique, à l'inverse d'une reconfiguration statique, implique une reconfiguration sans interruption de service.

Il a été souligné dans [27] que l'on pouvait distinguer deux types de reconfiguration dynamique : la reconfiguration dynamique avec dépendance de données et sans dépendance de données. La dépendance de données entraîne comme problématique la sauvegarde des données de traitement entre deux reconfigurations. L'aspect critique dans une reconfiguration dynamique est le temps de reconfiguration. Ce temps dépend de la taille du fichier de reconfiguration mais aussi de la vitesse de l'interface de communication. Lorsque l'application le permet (temps entre deux traitements consécutif) ou si l'on se place dans un contexte de reconfiguration statique on peut faire appel à une reconfiguration totale du circuit. Par contre si les contraintes de temps sont très strictes, on comprend qu'il est préférable de réduire au maximum le temps de reconfiguration du matériel et donc réduire la taille du fichier de reconfiguration. Ceci peut être fait grâce à l'utilisation de circuits permettant la reconfiguration d'une partie de ce circuit et non de sa totalité.

La technologie utilisée dans certains FPGA (notamment ceux de la famille Virtex de chez Xilinx) permet la reconfiguration d'une partie du composant sans perturber le fonctionnement de l'ensemble [28]. Ainsi, si plusieurs opérateurs sont instanciés sur un même FPGA, il est possible de venir reconfigurer l'un d'eux sans perturber le fonctionnement des autres. La conception d'un système permettant ce type de reconfiguration se fait par séparation en deux parties du système à instancier sur le FPGA. L'une appelée partie statique qui représente la partie non modifiée par un changement de contexte et l'autre est appelée partie reconfigurable. Le but étant de réduire au maximum la partie reconfigurable comme présenté dans [29]. On génère ainsi un fichier de reconfiguration pour FPGA (appelé bitstream) ne contenant que les informations relatives à la zone reconfigurable. Il est possible d'aller encore plus loin que cette technique appelée reconfiguration partielle en générant un bitstream ne contenant que les changements réels entre deux contextes d'exécution. Cette technique est appelée reconfiguration partielle par différence. Ce qui fait une économie non négligeable en termes de poids de fichier ce qui induit :

- une occupation mémoire réduite,
- un temps de reconfiguration réduit (moins de ressources à reconfigurer).

Le tableau suivant, illustrant un exemple de FIR (Finite Impulse Response) présenté en détail dans la partie 1.3.3 et optimisé pour une reconfiguration rapide, est fourni à titre d'exemple.

Il est néanmoins nécessaire de faire appel à un flot de conception spécifique et de définir une zone reconfigurable dynamiquement au sein du FPGA. Plus de détails sur ce type de matériel et sur les méthodologies de conception sont donnés en annexe du mémoire de thèse de J.P. Delahaye [30].

TAB. 1.1: *Comparaison reconfiguration totale / reconfiguration partielle*

Virtex-II 2000	Bitstream total	Bitstream partiel minimal
Taille du bitstream	830 ko	2 ko
Temps de configuration	16,6 ms	40 µs

Flexibilité logicielle : les processeurs logiciels

La flexibilité logicielle permet de faire abstraction des spécificités matérielles par l'utilisation d'un langage de programmation de haut niveau (type C/C++) et d'un compilateur dédié à l'architecture d'exécution. Elle permet aussi la modification d'une application s'exécutant sur une cible technologique par simple changement de code applicatif. Ce type de flexibilité est supporté par des architectures telles que les GPP, les DSP et ce que l'on appelle les processeurs logiciels (Microblaze [31] chez Xilinx ou Nios [32] chez Altera ou encore le processeur Leon [33]) qui peuvent être implantés sur des FPGA. L'introduction dans le domaine des équipements radio de la flexibilité logicielle est une évolution majeure à laquelle l'avènement de la radio logicielle a fortement contribué. En effet, les temps de conception et de validation dédiés aux circuits spécifiques font qu'il est extrêmement difficile de respecter les contraintes de temps de mise sur le marché d'un équipement. Le coût est également exhorbitant et ne peut être rentabilisé que dans le cas d'un succès pour un marché de masse. De plus lors de la conception, il faut s'assurer du bon fonctionnement de l'équipement dans de multiples environnements afin de limiter les taux de retours, il est donc nécessaire de bien concevoir le banc de test. Il est évident que si l'équipement possédait la capacité de pouvoir être modifié en cours de fonctionnement et ce sans remplacement d'aucune partie matérielle, les temps de mise sur le marché seraient drastiquement revus à la baisse. De plus si cette modification permettait une possible amélioration d'un ou plusieurs services cela deviendrait très avantageux à la fois du coté client et du coté constructeur (plus besoin de changer de matériel pour bénéficier des améliorations). Cette avancée est offerte par l'utilisation de logiciels au sein d'équipements radio. Il est évident que cette révolution à déjà commencé puisqu'il est possible pour les utilisateurs de télécharger dans leur équipement radio de nouveaux jeux, de nouvelles sonneries, de nouvelles applications, etc. Mais cela est loin de concerner la couche physique. Outre les contraintes technologiques qu'il faut lever, la régulation devra aussi faire sa révolution. La certification des équipements notamment en sera complètement remise en cause. Afin de profiter pleinement des apports de la radio logicielle pour améliorer la répartition dans l'attribution des ressources spectrales, la politique d'attribution des fréquences commence déjà à être modifiée comme en atteste [8] bien que ceci ne constitue que les premiers pas.

Dans un avenir plus lointain, il reste encore à franchir un pas vers le téléchargement de nouvelles normes radio et leur exécution au sein de l'équipement. C'est sur cette avancée majeure que repose le concept de la RL dans son sens le plus futuriste. Ainsi, un

équipement pourrait par exemple faire la demande de téléchargement d'une norme afin de permettre l'établissement d'une communication dans une certaine zone géographique. Il ne serait ainsi plus nécessaire de fournir à la conception la rétrocompatibilité avec les standards existants. De même qu'il ne serait plus nécessaire de prévoir des chaînes de conception d'équipements dédiées pour des zones géographiques précises. En effet, comme le montre la figure 1.5 un équipement possédant la flexibilité logicielle permet l'adaptation de l'équipement à l'évolution des différents standards radios existant et à venir. Ceci est vrai dans le cas où le matériel n'a aucune limitation. On le comprend bien, la flexibilité logicielle n'a pas vraiment de limites, au contraire du matériel sur lequel le logiciel s'exécute . On peut notamment nommer la limitation de mémoire disponible (pour le stockage des données mais aussi des fichiers de configuration téléchargés), la puissance de calcul disponible, la limite en consommation, le coût des composants, etc. La limitation de mémoire était un problème crucial il y a encore 5 ans dans un téléphone. Mais ce problème peut être considéré maintenant résolu grâce aux progrès fait dans ce domaine ces dernières années et dont le succès a fait baisser les prix significativement.

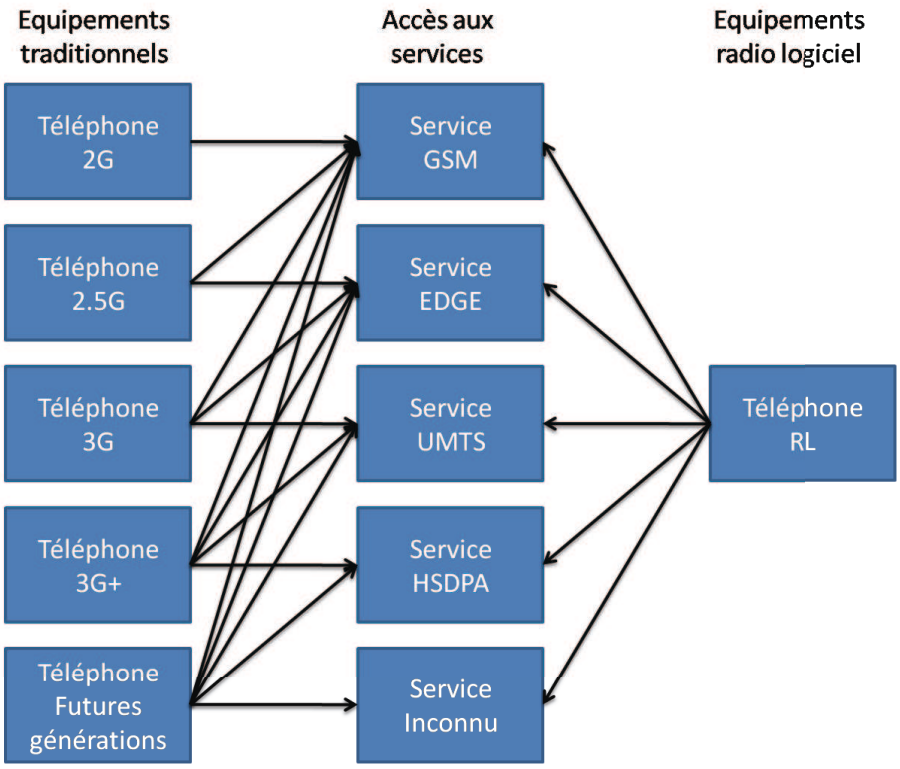


FIG. 1.5: *L'apport de la flexibilité logicielle*

Les concepteurs ne doivent cependant pas négliger les problèmes liés à cette approche qui implique l'utilisation massive de logiciel au sein d'équipements RL. En effet, de par la possibilité offerte de téléchargement permettant la modification d'organes importants du système, les risques pour l'intégrité de l'équipement sont réels et doivent faire l'objet d'études afin de minimiser ces risques et d'éviter les modifications non au-

torisées. Ce type de contrainte est particulièrement fort pour les systèmes dédiés aux domaines où les télécommunication se doivent d'être sûre (militaires, transports, ou encore sécurité pulblique). Ces risques sont multiples et déjà bien connus du monde de l'informatique tels que :

- bogues logiciels,
- incompatibilité entre divers logiciels,
- confidentialité (le système devient ouvert),
- introduction de virus informatiques par téléchargement,
- ou encore failles logicielles.

Il est évident que la façon de concevoir les futurs équipements radios va être fortement modifiée par la mise en place de services logiciels. Néanmoins on ne peut pas dire que les bases sont entièrement nouvelles. Ainsi, nous pouvons compter sur le savoir faire déjà présent des concepteurs de logiciels pour le monde informatique. En effet, il n'est plus besoin de démontrer la faisabilité de conception de logiciels s'exécutant sur un parc matériel hétérogène de par les ressources à disposition chez chaque utilisateur. Les défis sont cependant nombreux puisqu'il va falloir appliquer cette flexibilité logicielle dans le domaine embarqué où les contraintes sont élevées.

1.2.2 Composition hétérogène de la plateforme d'exécution

La problématique de fond, abordée en introduction, reste qu'il est peu probable de voir apparaître à court terme un équipement radio logicielle idéal. S'appuyant uniquement sur la flexibilité logicielle et les processeurs généralistes, cette approche est extrêmement consommatrice en énergie et reste improbable avec les technologies actuelles. Un contre exemple est la réalisation de BTS GSM à base de processeurs Pentium de la société VANU Inc, mais son faible volume de vente après une dizaine d'années de commercialisation peut susciter des interrogations quant à l'efficacité de la solution proposée. On s'oriente donc vers la conception d'équipements pour la RLR utilisant à la fois une flexibilité logicielle et une flexibilité matérielle. Cette démarche, comme présentée précédemment, nécessite l'utilisation de plateformes dites hétérogènes afin d'effectuer les différents traitements inhérents aux radiocommunications. Bien que cela ne soit pas le but de ce document de se pencher sur l'étude de telles plateformes, il est intéressant de les présenter afin d'appréhender la complexité de conception puisque notre contribution sera à intégrer dans le flot de conception d'équipements RC. La figure 1.6 montre une architecture RLR classique composée de trois étages de traitement.

Situé après l'antenne, l'étage RF, entièrement réalisé en analogique, permet de faire l'acquisition du signal capté par l'antenne et d'en sélectionner la partie désirée en jouant le rôle de filtre passe bande. Une amplification faible bruit de ce signal est ensuite effectuée avant traitement par la partie suivante. Ensuite, un échantillonnage en fréquence

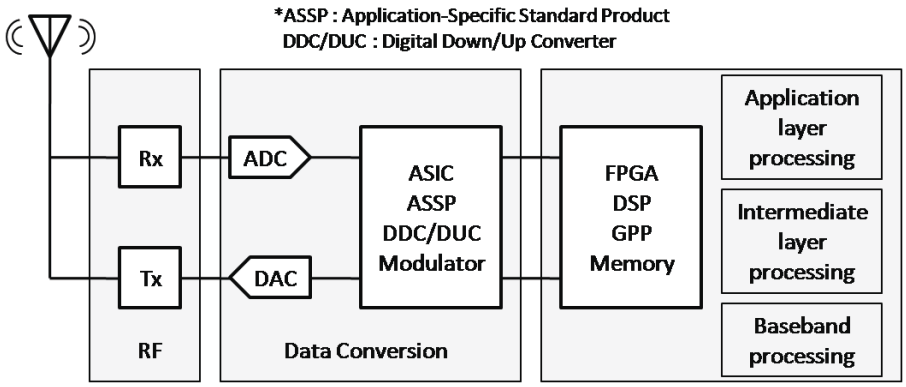


FIG. 1.6: *Plateforme SDR classique*

intermédiaire (FI) sera réalisé dans la partie "Data Conversion". L'étage FI a pour but d'améliorer la sélectivité (par des filtres dont les flancs sont le plus raide possible de façon à rejeter au mieux les résidus des canaux adjacents) et la sensibilité (par une amplification la plus linéaire possible afin de garder un gain constant) du récepteur. La troisième partie est la partie appelée traitement en bande de base qui est réalisée à l'aide de circuits numériques et qui est donc en grande partie reconfigurable afin de s'adapter aux différents traitements. Cette partie effectuant le traitement en bande de base regroupe les opérations telles qu'estiamation de canal, égalisation, synchronisation décodage canal, etc. On peut également regrouper avec les traitements de la couche physique ceux des couches protocolaires supérieures et notamment la couche applicative. Elle peut impliquer du traitement du signal très exigeant pour le traitement video par exemple. Ce rapprochement de traitements permet d'envisager des optimisations inter-couches également (*cross-layer design*).

Finalement, il apparaît qu'une plateforme pour la RLR se compose d'une partie analogique et d'une partie numérique. Un concepteur doit donc pouvoir interfacer ces deux mondes et concevoir des systèmes performants pouvant être reconfigurés suivant les besoins. Avec une telle complexité et hétérogénéité, on comprend que les procédés permettant d'effectuer les reconfigurations ne seront pas aisés à mettre en place. En effet, lors d'une reconfiguration il va falloir pouvoir acheminer les données de reconfiguration afin de cibler les ressources se devant d'être reconfigurées. Ceci doit être effectué en prenant en compte les différences de temps de traitement entre chaque circuit utilisé, de même que la prise en compte du temps effectif de reconfiguration pour ne pas perturber le traitement en cours.

1.2.3 Exemples de projets “historiques” de la RL

Nous venons de présenter le type de ressources nécessaires à la conception de systèmes pour la RLR. Nous avons aussi défini les difficultés de mise en œuvre et d'utilisation de plateformes hétérogènes. Nous allons maintenant présenter deux projets “historiques” de la RL qui font référence : SpeakEasy [34] et SpectrumWare [35].

SpeakEasy

Débuté en 1991, le projet militaire SpeakEasy, fut le premier projet pour la radio logicielle. Il avait été initié afin de palier les problèmes d'incompatibilités entre les moyens de communication des différentes branches de l'armée américaine. Ce projet était très ambitieux puisqu'il avait pour objectif de permettre l'établissement de 10 formes d'ondes différentes à partir d'un seul équipement. En 1994, une démonstration fut faite qui révéla la réussite du projet. Le problème était l'encombrement de l'équipement puisque celui-ci occupait tout l'arrière d'un camion et sa complexité de conception (il intégrait plusieurs centaines de processeurs). Il a été conçu pour être un système multibandes dans une gamme de 2 à 400 MHz. Les leçons qui ont été tirées de ce prototype furent qu'il faut apporter un soin particulier à l'établissement d'un environnement ouvert afin de permettre le développement rapide de nouvelles applications. En effet, le logiciel s'exécutant sur cette plateforme était codé en code assembleur spécifique au matériel utilisé, ce qui était donc loin d'offrir une portabilité étendue. Un autre point soulevé est qu'il était “surdimensionné” on peut comprendre par là que le prototype devait manquer de flexibilité afin de s'adapter à de futures contraintes. De plus, l'établissement d'un tel équipement a été extrêmement coûteux puisqu'il était composé des produits les plus avancés disponibles sur le marché à l'époque.

SpectrumWare

SpectrumWare est un projet de recherche du MIT débuté en 1994 et qui a pour but de révéler le potentiel d'implantation portables (au sens de l'implantation sur plusieurs plateformes) pour la radio logiciel. L'architecture retenue était composée d'un front-end mettant en œuvre la partie RF et la conversion analogique/numérique et numérique/analogique. La partie traitement était implémentée sur un PC sous Linux. La différence notable avec le projet SpeakEasy était l'attention portée sur la portabilité du logiciel et sa réutilisation en utilisant un système d'exploitation (OS pour Operating System) dédié et une conception basée sur une approche logicielle orientée objet. Ainsi le choix du matériel à utiliser découlait du logiciel utilisé. Ce qui est ressorti de cette étude fut que bien que n'étant pas la plateforme la plus puissante du moment, en effet, l'utilisation d'un OS et une conception orientée objet induisent indubitablement des latences (ou

overhead), la progression en terme de puissance de calcul, d'espace de stockage et de rapidité d'exécution des ressources matérielles dans le futur la feront devenir beaucoup plus attractive.

Une évolution sera portée à ce prototype en 1998 par une séparation du signal de traitement et du signal de contrôle et de configuration. Ce qui a pour but d'améliorer encore la portabilité et la réutilisabilité logicielle.

1.2.4 Gestion logicielle des ressources hétérogènes

Cette nouvelle approche qu'est la RLR introduit on l'a vu, une forte utilisation de logiciels au sein de la plateforme d'exécution. Il va donc falloir mettre en place une interface entre la couche logicielle et la couche physique afin de permettre le contrôle et la reconfiguration des ressources. Ce type d'approche est illustré par la figure suivante qui extraite de [30].

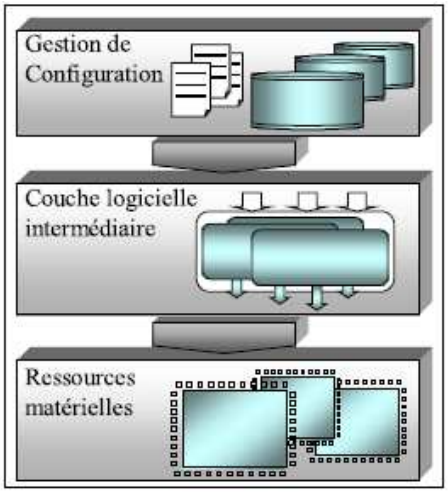


FIG. 1.7: *Gestion logicielle des ressources hétérogènes*

Vu l'hétérogénéité du parc d'équipement, il est nécessaire de développer des logiciels pouvant s'exécuter sur des plateformes différentes. Il faut donc fournir à chaque équipement un moyen d'interpréter les instructions du logiciel et de permettre le routage des informations (données et contrôle de reconfiguration) jusqu'aux ressources matérielles spécifiques. Ceci est réalisé grâce à une couche logicielle intermédiaire qui permet un interfaçage entre le monde logiciel et matériel. De plus il est nécessaire de proposer une architecture logicielle ouverte, des interfaces de programmation standardisées (ou API pour *Application Programming Interface*) et permettre la réutilisation des modules de traitement dans différents contextes d'exécution. Il y a néanmoins un coût non négligeable à payer pour l'utilisation d'une telle couche. Ce surcoût s'applique au niveau de l'occupation mémoire, aux latences engendrées pour l'acheminement des données ou

encore à l'accroissement de la consommation. Ceci est néanmoins le prix à payer afin de permettre la conception d'un équipement pouvant exécuter un ensemble d'applications permettant l'établissement d'une quelconque communication radio par reconfiguration matérielle. Au-delà de cette couche logicielle, il va aussi falloir prendre en compte la granularité de reconfiguration offerte à l'équipement. En effet, on peut distinguer deux types de reconfigurations matérielles : la reconfiguration gros grain, et la reconfiguration grain fin. Plus la granularité est élevée et plus le nombre de composants reconfigurés est important mais plus la flexibilité diminue. Il va donc falloir prendre en compte dans la conception le fait que des circuits à granularités différentes cœxistent au sein du même équipement afin d'organiser au mieux le chargement des codes et des configurations.

Nous allons présenter la proposition qui fait référence au niveau international en tant que couche logicielle aidant à la conception de systèmes pour la RLR : le SCA [36].

Software Communication Architecture

Initié par le *Department of Defence* (DoD) américain dans son projet *Joint Tactical Radio System* (JTRS), ce projet a pour but d'aider la conception de systèmes pour la RLR. Actuellement repris par le SDR Forum afin d'en assurer l'évolution et la promotion, le SCA est devenu incontournable pour la conception des futurs systèmes. En effet, il est devenu une condition nécessaire pour tout équipement de communication acheté par le DoD. Son architecture est composée d'une couche d'application et d'une couche appelée *Operating Environment*. Cette dernière est composée d'un OS temps réel (RTOS pour *Real Time Operating System*), d'une couche de communication et d'un *Core Framework* regroupant les descriptions des informations de configuration matérielle et logicielle de la plateforme. Afin de maximiser l'interopérabilité, le SCA est basé sur des standards commerciaux et donc déjà éprouvés tels que CORBA [37] (*Common Object Request Broker Architecture*), POSIX [38] (acronyme de *Portable Operating System Interface*) ou encore CCM [39] (*Corba Component Model*). C'est ce qui en fait à la fois sa force et sa faiblesse. En effet, étant basé sur CORBA, qui est connu historiquement pour ne pas gérer nativement les circuits spécialisés type DSP ou FPGA, il ne supporte que les plateformes multiprocesseurs généralistes. Il est donc nécessaire de recourir à l'utilisation de modules externes à CORBA afin de les décrire. De fait, cela rajoute un overhead sur les communications. Il existe des travaux [40] permettant d'accélérer les connexions comme des adaptateurs réalisés sur GPP ou l'ajout de bus rapide. La difficulté de faire reconnaître un FPGA comme un composant SCA est principalement due au fait que la programmation de ce type de circuit est réalisée à très bas niveau. De plus, cette programmation est fortement liée aux ressources spécifiques à chaque gamme de FPGA et il n'existe pas d'équivalent à l'adressage virtuel fournit par les OS. Malgré cela il est possible de définir un certain nombre de fonctionnalités au FPGA pour le faire reconnaître comme composant du SCA et bénéficier de ces possibilités de reconfiguration. Généralement cette reconnaissance est obtenue par la mise en place d'une couche

d'abstraction, codée par le concepteur lui-même, entre les interfaces de communication et le module de traitement. Ce qui a pour effet d'introduire comme dit précédemment un overhead non négligeable. Jusqu'à récemment, il était possible d'intégrer CORBA au sein d'un FPGA en recourant à l'utilisation d'un processeur logiciel. Le problème majeur est que ces solutions limitent la transparence et l'efficacité de la mobilité des applications, de plus ce type de solution doit être entièrement conçue par le développeur, ce qui limite la réutilisabilité de la solution. Récemment, une solution appelée *ORBexpress FPGA* d'Objective Interface[3] a vu le jour. Décrite en langage VHDL, cette entité peut être directement instanciée dans un FPGA lui permettant ainsi d'être reconnue comme composant SCA et donc permettre des migrations d'applications entre GPP, DSP ou FPGA de manière transparente.

Il y a deux manières de masquer l'hétérogénéité. Soit utiliser une abstraction générique du matériel (tel que l'utilisation d'une machine virtuelle Java) commune à tout matériel. C'est le choix CORBA du SCA. Mais alors, les capacités particulières à chaque type de matériel ne sont pas exploitées à tout leur potentiel. Soit on effectue un adaptateur sur mesure à chaque matériel, afin d'explorer au mieux ses capacités intrasèques. C'est notre approche.

Après avoir exposé sommairement quelques projets faisant référence sur la problématique de la RLR, nous allons présenter une architecture logicielle initialement développée par J.P. Delahaye au sein de l'équipe SCEE et à laquelle j'ai contribué. Les objectifs de ces travaux n'étaient pas aussi vastes que ceux visés par le SCA mais ciblaient les particularités des composants constituant la plateforme d'exécution. Ainsi, si le SCA permet une aide à la conception et au déploiement d'une chaîne radio, il ne permet pas une prise en compte des différentes possibilités de reconfiguration des composants d'une chaîne déjà déployée et en cours d'exécution. Ce service est l'un des attraits de l'architecture logicielle présentée ci-après.

1.3 Une architecture logicielle de gestion de reconfiguration : le HDReM

1.3.1 Présentation

Afin de faire fonctionner un standard de communication, il faut déployer un certain nombre de traitements. Il est possible de décomposer un standard en un ensemble de fonctions elles-mêmes constituées de divers blocs de traitement que l'on nomme opérateurs. Si l'on regarde de plus près l'ensemble des fonctions utilisées dans l'établissement des divers standards actuels, on remarque des similarités entre certaines d'entre elles.

[3] voir "CORBA for FPGAs : Tying together GPPs, DSPs, and FPGAs" dans www.dsp-fpga.com

Des travaux [41] ont mis en évidence l'intérêt de la réutilisation de blocs de traitement afin de faciliter la cœxistence de différents standards sur une même plateforme. De là est née l'approche par "opérateurs communs" [42] pour l'établissement de systèmes de communication multi-standards. Notre approche, pour la conception d'équipements de communication dans un contexte radio logicielle, utilise ce concept d'opérateurs communs reconfigurables s'adaptant aux contraintes du standard à utiliser, même si elle ne s'y limite pas.

J.P. Delahaye a défini dans son mémoire de thèse [30] lors de ses travaux au sein de l'équipe SCEE, une architecture de gestion de reconfiguration appelée HDReM (Hierarchical and Distributed Reconfiguration Management) adaptée à un contexte RLR. La dernière année de sa thèse concordant avec ma première année j'ai pu assimiler ses travaux qui m'ont servis de support dans mon travail. HDReM propose d'ajouter une surcouche de gestion de la reconfiguration des opérateurs à la chaîne classique d'opérateurs effectuant les opérations de traitement du signal radio. Ceci prend tout son sens dans un équipement de radio logicielle où la flexibilité de l'implantation des traitements est poussée au maximum des capacités du matériel. L'un des points remarquable de cette architecture est la séparation fonctionnelle, et physique si on le souhaite, du chemin de données (entre chaque opérateurs participant à l'établissement d'une communication) et du chemin de reconfiguration (entre chaque entité de gestion de reconfiguration de l'architecture HDReM) comme illustrée par la figure 1.8. Ceci implique également une conception de l'application RL séparant les fonctions radio elles mêmes des fonctions de gestion de la reconfiguration. En effet, si l'on suppose que chaque opérateur de la chaîne de traitement radio est reconfigurable alors chacun de ses opérateurs possède des données de reconfiguration qui lui sont propres. De plus, ces données dépendent de la nature de cet opérateur mais aussi de sa cible matérielle d'exécution. Dans une architecture ne présentant pas de séparation de ces chemins, il faut encapsuler les données dans un format permettant à chaque opérateur de reconnaître une donnée qui lui est destinée ou non (qu'il va traiter ou non donc) puis une en tête lui permettant d'identifier une donnée à traiter ou une donnée de reconfiguration. Cette opération oblige donc à identifier l'en tête de chaque paquet de données transmis à chaque opérateur. De plus chaque opérateur doit pouvoir remettre en forme ce paquet afin de le transmettre au prochain opérateur. On comprend donc que ce type de solution entraîne un overhead important dans le temps de traitement comme dans le temps de reconfiguration du système. Nous proposons avec cette architecture HDReM de diminuer le surcoût des éléments permettant la lecture et l'écriture de ces entêtes ainsi que la diminution de l'overhead induit par cette encapsulation par une séparation des chemins de données et de reconfiguration afin de permettre l'adressage de manière ciblée de chaque éléments de traitement possédant des facultées de reconfiguration.

Un autre point important est que la spécialisation des gestionnaires de reconfiguration permet de prendre en compte les spécificitées de l'opérateur de traitement ainsi que les contraintes relatives à la cible matérielle d'exécution. Nous voyons donc sur la figure précédente que chaque opérateur est controlé par un Reconfiguration Manager spécifique

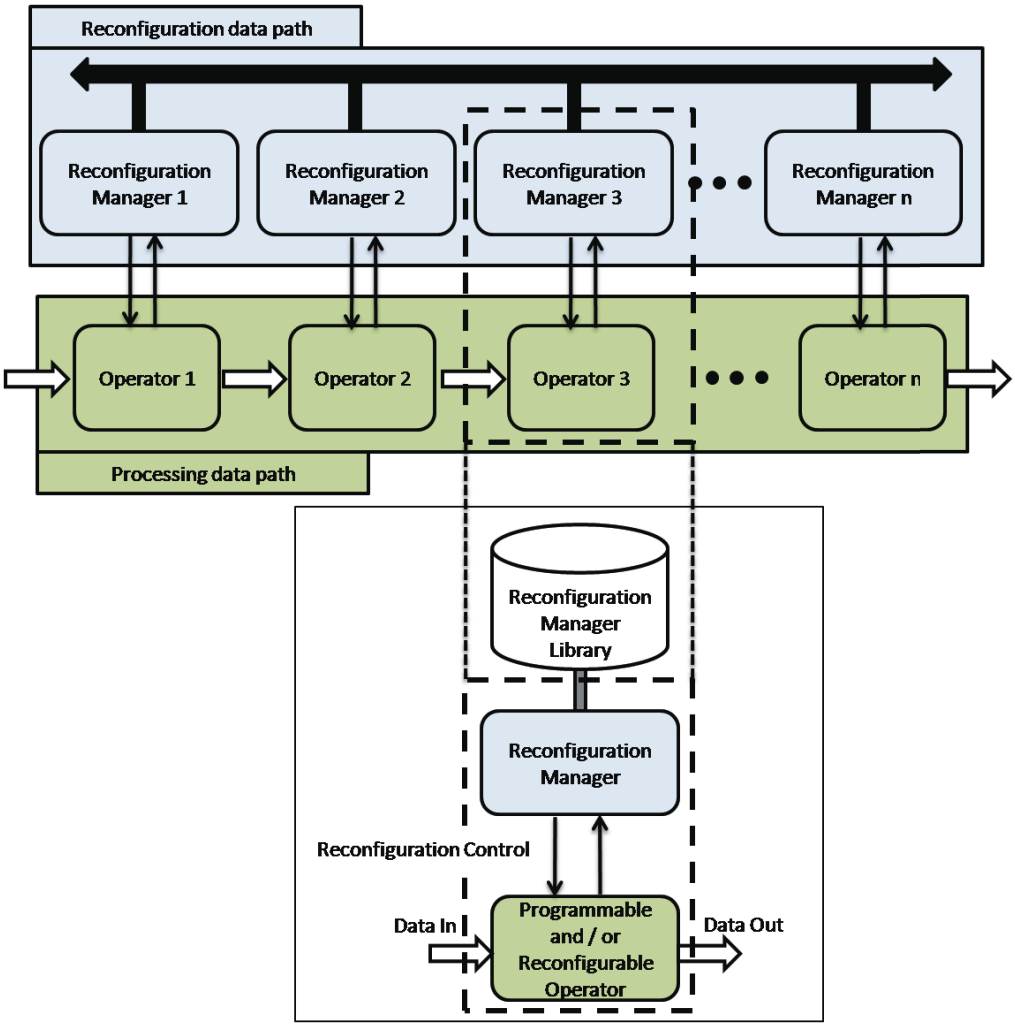


FIG. 1.8: *Séparation du chemin de traitement et du chemin de reconfiguration*

(ou ReM). Chaque entité ReM est donc optimisée pour un type de ressource reconfigurable. Ce type de gestion est appelée gestion distribuée. Il reste néanmoins difficle de gérer l'ensemble d'un système radio par une unique gestion distribuée en raison du nombre important de ressources et de l'hétérogénéité de la platefome d'exécution. Ainsi nous avons mis en place une structure hiérarchique de cette architecture afin de centraliser les reconfigurations d'un ensemble de ressources et de permettre une connaissance globale des paramètres du système.

Il y a été défini une distribution sur trois niveaux de hiérarchie en forme pyramidale comme présenté par la figure 1.9.

Les opérateurs présents en bas de cette figure constituent la chaîne d'opérateurs contenue dans un équipement classique. On voit donc qu'afin de gérer à 100% la flexibilité offerte par la reconfiguration il faut rajouter l'ensemble de l'architecture de gestion

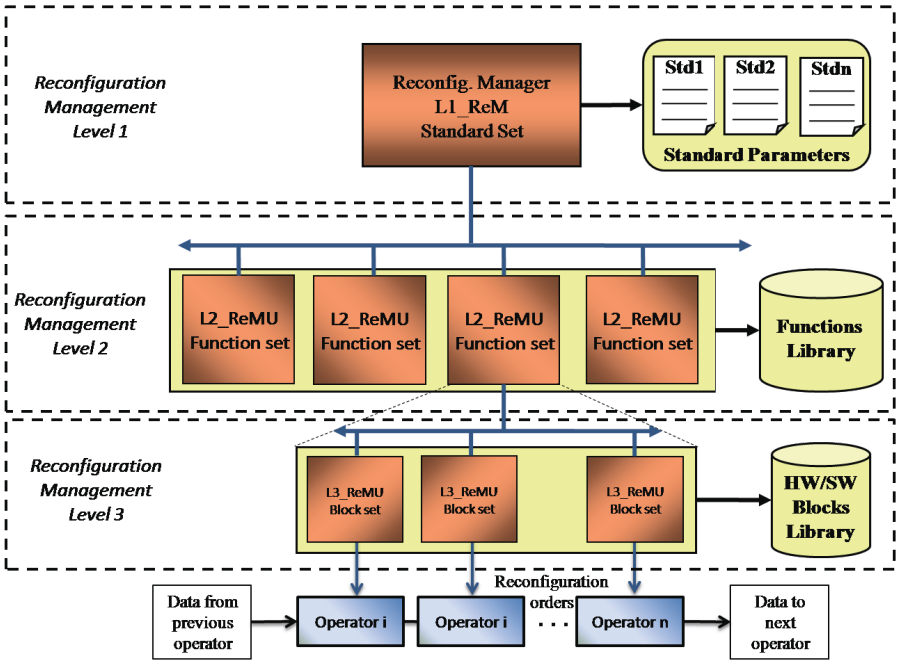


FIG. 1.9: *Architecture HDReM*

qui est au-dessus. Cela peut sembler de prime abord rajouter une complexité de gestion ainsi qu'une occupation accentuée en termes de ressource, néanmoins, c'est un petit prix à payer en comparaison des avantages attendus.

Cette architecture se définit ainsi : au niveau de plus haute hiérarchie se situe un unique gestionnaire de reconfiguration appelée L1_ReM qui a un rôle de superviseur général de reconfiguration. Cette entité logicielle est complètement indépendante du matériel dans son fonctionnement, il est important de noter que le module matériel sur lequel cette entité sera implantée deviendra le module maître de la plateforme. Cette entité s'occupe de l'instanciation et du contrôle d'une ou de plusieurs unités de niveau inférieur appelées L2_ReMU. Chacune de ces unités a pour fonction l'instanciation et le contrôle d'une ou plusieurs unités de niveau inférieur appelées L3_ReMU. Situées au niveau hiéarchique le plus bas, chaque unité L3_ReMU gère un opérateur permettant la réalisation d'une partie du traitement global du standard de communication. On voit donc que les données de reconfiguration s'échangent de l'entité L1_ReM vers une ou plusieur entitées L3_ReMU ce qui fait de l'architecture HDReM une architecture de type Top/Down pour l'envoi d'informations de reconfiguration.

Le L1_ReM est interfacé avec l'extérieur et prend en charge les demandes de changement de context venant des couches supérieures (demande de l'utilisateur ou du réseau par exemple). Ces demandes sont interprétées par le L1_ReM qui va cibler le ou les gestionnaires, de niveaux inférieurs, devant être reconfigurés et acheminer les données de reconfiguration jusqu'aux opérateurs de traitement du signal devant être reconfigurés.

Pour ce faire, l'entité L1_ReM envoie aux niveaux de gestion inférieurs un ensemble de paramètres contenant les fonctionnalités à reconfigurer (cf. figure 1.10).

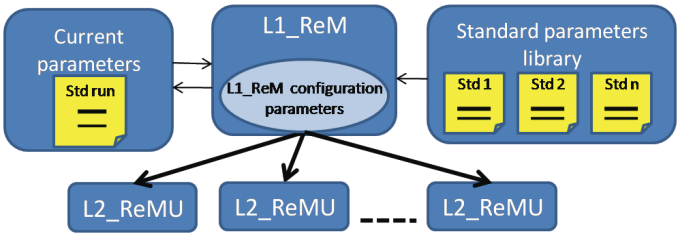


FIG. 1.10: *Niveau de gestion de configuration L1_REM*

Il est à noter que les informations envoyées par le L1_ReM aux différents L2_ReMU sont d'ordre abstrait. "Abstrait" est utilisé ici dans le sens où ces informations de configuration ne tiennent pas compte des contraintes apportées par les différentes cibles technologiques utilisées sur la plateforme. Le L1_ReM possède la connaissance de la répartition de ses L2_ReMU au sein du système ainsi que les moyens de communication à utiliser pour échanger avec eux. L'entité L1_ReM est en charge du contrôle des paramètres des différents standards et de la mise à jour des paramètres courants. Ceci est permis par une connexion avec une bibliothèque renseignant sur l'étendue des possibilitées de reconfiguration de la plateforme. Une partie des informations de cette bibliothèque renseigne le L1_ReM sur les configurations de paramètres possibles (appelée Standard parameters library, à droite sur la figure 1.10). Une seconde partie renseigne sur l'état actuel de chaque L2_ReMU implantés (appelée Current parameters, à gauche sur la figure 1.10). Ce qui lui permet de cibler précisément le ou les l2_ReMU à reconfigurer.

Les unités de gestion de reconfiguration de niveau 2 (ou L2_ReMU) sont en charge de la mise en place des fonctions de traitement dans la chaîne de transmission. Ces entités sont directement interfacées avec le L1_ReM afin de recevoir les ordres de reconfiguration ainsi que les paramètres de reconfiguration. Le ou les L2_ReMU recevant les instructions venant de l'entité L1_ReM interprètent ces données et se reconfigurent suivant les besoins. Chaque L2_ReMU, ciblé par le L1_ReM pour une nouvelle configuration, détermine pour la fonction qu'il doit réaliser quelle est la configuration de traitement la mieux adaptée. Ceci dépend principalement des débits requis et des ressources physiques disponibles. Une fois la configuration de traitement définie, chaque L2_ReMU adresse les L3_ReMU nécessaires à la bonne réalisation de la fonction (cf. Figure 1.11). Comme pour l'entité L1_ReM, chaque L2_ReMU est interfacé avec une bibliothèque renseignant à la fois sur les possibilités de reconfiguration des paramètres pour la fonction dont il a la charge ainsi que sur l'état actuel des unités L3_ReMU dont il a la charge.

Les unités de gestion de reconfiguration de niveau 3 (ou L3_ReMU) sont regroupées sous le contrôle de différents L2_ReMU. En effet, nous rappelons qu'un L2_ReMU est en charge de la gestion de reconfiguration d'une fonction et qu'une fonction est composée de plusieurs sous blocs de traitement (ou opérateur). Il est donc naturel d'asso-

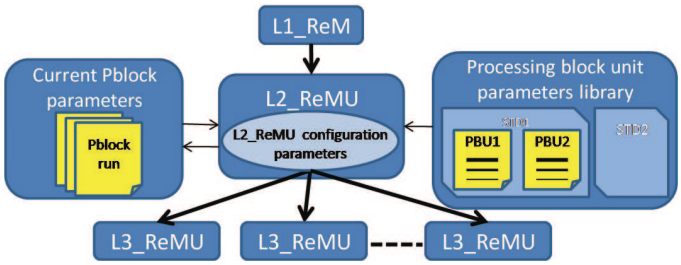


FIG. 1.11: *Niveau de gestion de configuration L2_REMU*

cier à chaque opérateur un gestionnaire de reconfiguration de niveau 3 et que plusieurs L3_ReMU soient controlés par un même L2_ReMU. Chaque L3_ReMU est responsable d'un unique opérateur et doit le reconfigurer suivant les indications de son L2_ReMU. Un L3_ReMU a aussi la charge d'implanter l'opérateur dont il a la responsabilité sur une cible matérielle (cf. Figure 1.12).

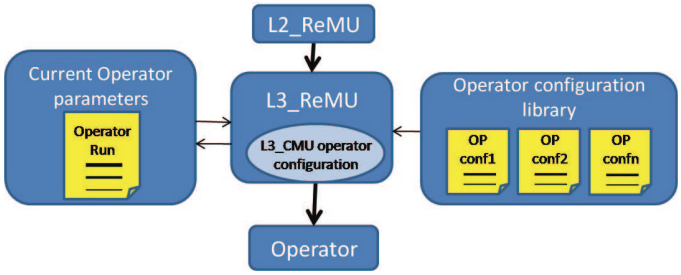


FIG. 1.12: *Niveau de gestion de configuration L3_REMU*

Ainsi sa fonction est de reconfigurer les ressources matérielles en vue de l'exécution de l'élément de traitement paramétré dont il a la charge. Il est à noter que la nature des données de reconfiguration manipulées par un gestionnaire de niveau 3 dépend fortement du type de ressources utilisées. Chaque L3_ReMU est interfacé avec une bibliothèque contenant les fichiers exécutables de l'opérateur dont il a la charge (fichier binaires ou bitstreams par exemple) afin de réaliser une nouvelle implantation en cas de reconfiguration. De même, la bibliothèque renseigne aussi sur l'état courant de l'opérateur ainsi que son mode de fonctionnement.

Une particularité de cette architecture est que l'on peut voir le niveau de gestion de reconfiguration 2 comme une interface entre une gestion de haut niveau sous forme de paramètres effectuée par le niveau L1_ReM et la gestion de reconfiguration des ressources matérielles effectuée par les unités du niveau 3. En effet, les données de reconfiguration fournies par l'entité L1_ReM sont complètement indépendantes des ressources matérielles présentes sur la plateforme d'exécution. À l'inverse, les données de reconfiguration fournies par les unités de gestion de reconfiguration du niveau 3 sont complètement dépendantes des cibles matérielles d'exécution. De plus, cette distribution nous permet de traiter des reconfigurations :

- à l'échelle d'un changement complet de standard (qui sera pris en charge par le L1_ReM),
- de granularité plus fine avec le changement (ou mise à jour) d'une fonction assurant la meilleure QoS pour le standard choisi,
- ou encore intervenir très précisément pour une reconfiguration locale d'un opérateur implanté sur une quelconque cible technologique déployée sur la plateforme.

Un autre point important est que cette architecture (grâce aux gestionnaires de bas niveaux L3_ReMU) peut s'adapter à tous les contextes et est particulièrement adaptée à l'utilisation de FPGA au sein de la plateforme. Cela permet de pouvoir tirer pleinement parti de la reconfiguration partielle qui est une technologie en devenir pour la radio logicielle.

Si l'on regarde le cas du scénario de reconfiguration le plus exigeant (i.e. une reconfiguration totale des diverses éléments constituant la chaîne de traitement), on se retrouve dans un changement de contexte de niveau de granularité le plus important. Autrement dit, les parmatètres de reconfiguration courants sont différents des futurs paramètres. Dans ce cas, les nouveaux paramètres sont envoyés à chaque unité de reconfiguration afin que l'ensemble des éléments soient reconfigurés. Lors d'une telle reconfiguration, on peut s'interroger sur la pertinence d'une telle architecture. En effet, dans le cas d'une gestion centralisée, un seul ordre de reconfiguration globale permet la reconfiguration de la totalité de la chaîne de traitement. Néanmoins, la gestion hiérarchique permet de cibler individuellement les éléments de traitement. L'approche hiérarchique est d'une part approriée pour gérer divers types de données de reconfiguration de granularité multiple. D'autre part ce type de gestion facilite le contrôle de la reconfiguration partielle de la chaîne de traitement et permet donc de contrôler des granularités très fine de reconfiguration. De plus cette architecture gère aussi le flot de données pendant les phases de reconfiguration ce qui est primordial afin d'assurer la continuité de service malgré la reconfiguration. Finalement, cette gestion rend possible le contrôle de chaque élément présentant des possibilités de reconfiguration et accroît ainsi la flexibilité de l'ensemble. On peut aussi souligner que la solution choisie de créer une architecture distribuée pour le chemin des données de reconfiguration permet de répartir les tâches de gestion de reconfiguration des ressources disponible du système sans surcoût de contrôle important localisé sur une seule ressource. Ceci est bénéfique en comparaison d'une solution centralisée qui nécessite un processeur dédié à cette tâche pour son exécution.

Il est possible de classer les besoins de reconfiguration en deux catégories, les reconfigurations inter-standard et les reconfigurations intra-standard. Il est évident que les reconfigurations de type inter-standard requerront la reconfiguration d'un nombre plus important (voire la totalité) d'éléments de traitement. Ainsi, suivant les contraintes de temps sur la reconfiguration il faudra prévoir la possibilité de créer une deuxième chaîne de traitement en parallèle de celle en cours d'utilisation. En effet, si il n'y a pas de contrainte de temps et que le changement de standard s'effectue lors de la non utilisation des communications : il est alors possible de détruire les éléments de l'ancienne

chaîne de traitement en faveur d'une nouvelle supportant le nouveau standard de communication en vigueur dans la proximité de l'équipement. A contrario si l'équipement se déplace dans une zone nécessitant le changement de standard en cours d'utilisation : il va faloir pouvoir mettre en place une deuxième chaîne de traitement en parallèle de la première et effectuer une commutation lorsque cette dernière sera opérationnelle. Libre alors de détruire l'ancienne chaîne de traitement ou de la garder effective pour une future utilisation. Dans le cas de reconfigurations intra-standard, les reconfigurations effectuées sur les éléments de la chaîne de traitement seront le reflet des conditions d'utilisation de l'équipement (déplacement intérieur/extérieur, qualité de la réception, niveau de batterie, etc.) et devront donc pouvoir être adaptées le plus rapidement et le plus efficacement possible.

On peut synthétiser l'apport de l'architecture HDReM comme suit :

- Possibilité de cibler directement et individuellement les éléments à reconfigurer par une séparation des chemins de données et des chemins de reconfiguration.
- Introduction d'une gestion de données de reconfiguration de différentes natures par la mise en place de niveaux hiérarchisés.
- Les trois niveaux hiérarchiques ont été définis en correspondance avec les besoins en terme de reconfigurabilité des applications.
- Les reconfigurations peuvent mettre en jeu un nombre variable d'éléments à reconfigurer. La reconfiguration hiérarchique distribuée permet de répondre aux besoins de gestion de granularité multiple de reconfiguration.
- La gestion hiérarchique permet une connaissance globale de l'état d'un ensemble d'éléments de traitement de niveau inférieur et permet une prise globale de décision de reconfiguration.

1.3.2 Opérateurs de traitement

Les opérateurs de traitement exécutant les tâches permettant la réalisation d'un standard de communication sont déployés sur une plateforme hétérogène. Cette hétérogénéité induit une différence de modes de traitement (séquentielle dans les processeurs, parallèle dans les FPGA ou les ASIC par exemple) ainsi que des différences d'interface. Il est donc essentiel de définir un composant possédant une interface standardisée, encapsulant l'opérateur de traitement, permettant un contrôle simple et une réutilisabilité étendue des opérateurs par l'architecture de gestion. Notre choix s'est porté sur l'utilisation de mécanismes de transmissions de type asynchrone entre les composants. D'une part, ceci est le mode de fonctionnement des processeurs qui exécutent des fonctions à un rythme indépendant de l'arrivée réelle des données. D'autre part, ce mode de fonctionnement permet de s'abstraire des problème de synchronisation dûs aux différences de cadence de fonctionnement entre circuits et permet donc une indépendance des données vis-à-vis de la communication entre les opérateurs. Ceci à pour double intérêt de

faciliter l'interconnection de fonctions matérielles rythmées différemment mais aussi de fonctions exécutées sur processeurs avec d'autres implantées sur ASIC ou FPGA. Les opérateurs encapsulés dans ces composants ont un fonctionnement synchrone en interne. Au final, l'ensemble de la chaîne de traitement aura un fonctionnement d'exécution de type GALS (*Globally Asynchronous Locally Synchronous*) [43] sur l'ensemble des ressources disponible sur la plateforme d'exécution. Dans ce type d'exécution, chaque composant a pour fonction de contrôler la production/consommation de données vis-à-vis de la consommation/production des composants auxquels il est connecté. Ainsi, comme chaque traitement se termine par la production de données, le composant doit vérifier qu'il est possible d'en faire l'envoie au composant suivant. La figure 1.13 suivante, extraite de [30], illustre un élément de traitement encapsulé dans un composant de type GALS.

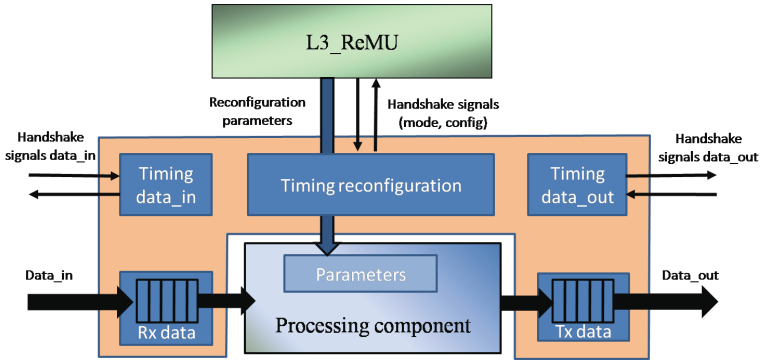


FIG. 1.13: *Élement de traitement encapsulé dans un composant de type GALS*

Sur cette figure on peut voir les deux interfaces de communication permettant à chaque composant de s'inteposer et se synchroniser avec les autres. En effet, l'état des fifos, stockant un ensemble de données, présentent en entrée et en sortie du composant (*Rx data* et *Tx data*) permet de signaler le début ou l'arrêt d'un traitement vis-à-vis des données à produire ou à consommer. Les signaux de contrôle du flot de données asynchrone sont gérées par les blocs *timing data_in* et *timing data_out*. Le bloc *Timing reconf* renseigne sur le mode de fonctionnement (initialisation, traitement, transparent, erreur).

1.3.3 Exemples de réalisations

Nous allons présenter succintement dans cette section trois exemples de mise en œuvre de l'architecture HDReM. Tout d'abord nous montrerons comment un gestionnaire de reconfiguration peut tirer partie des spécificités matérielles en effectuant une reconfiguration sur une partie d'un composant matériel sans bloquer son fonctionnement. Puis nous présenterons un opérateur de traitement réalisé suivant l'approche opérateur commun avec son gestionnaire de reconfiguration de niveau 3. Enfin nous présenterons

succinctement une réalisation d'un système basé sur une plateforme multiprocesseurs (une version plus détaillée pourra être consultée dans [44]).

Matrice d'interconnexion reconfigurable

Comme dit précédemment, l'architecture HDReM permet un contrôle non seulement des opérateurs de traitements mais aussi des chemins de données les reliant les uns aux autres au sein de la plateforme matérielle d'exécution. Cette liberté laissée sur la reconfiguration des chemins de données permet d'envisager des reconfigurations à la volée (ou reconfiguration *on-the-fly*) [45]. En effet, la création d'un nouveau bloc de traitement en parallèle du premier permet de ne pas interrompre le traitement jusqu'à ce que l'élément requis soit créé. Dès lors, un reroutage des données permet de basculer directement de l'ancienne configuration à la nouvelle sans interrompre le traitement, libre ensuite de supprimer l'ancienne ressource ou de la garder pour une future réutilisation. Plusieurs approches sont possibles afin de réaliser une telle connexion on peut par exemple nommer les réseaux sur puce (appelé *Network on Chip* ou NoC) ou encore les matrices d'interconnexion. La figure 1.14 illustre une reconfiguration de chemin de données à l'aide d'une matrice d'interconnexion reconfigurable présentée dans [46]. La reconfiguration illustrée ici est utilisée en vue de déconnecter un opérateur de la chaîne de traitement, sans interruption dans le traitement des données, en tirant partie de l'ordonnancement efficace des tâches d'exécution en cours. Ce travail d'ordonnancement et de gestion des chemins de données est contrôlé par un L2_ReMU.

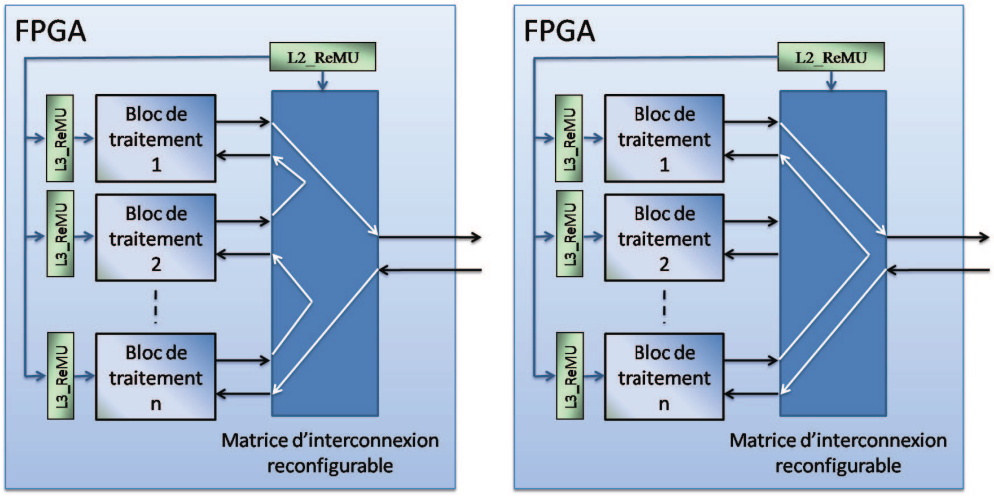


FIG. 1.14: *Matrice d'interconnexion reconfigurable*

Réalisation d'un opérateur commun et de son gestionnaire de reconfiguration de niveau 3

L'opérateur de traitement présenté ici est un filtre FIR, présenté par J.P. Delahaye, ayant obtenu le premier prix dans la catégorie des projets étudiants doctorants du concours national de conception organisé par le CNFM en partenariat avec la société Xilinx. Bien que ce filtre ait été conçu lors des travaux de thèse de J.P. Delahaye, j'y ais personnellement contribué sous la forme d'aide à la conception.

Un filtre FIR réalise la fonction de filtrage de mise en forme spécifiée dans de nombreux standards de radio communications tel que l'UMTS. L'approche vise à intégrer ce filtre à une architecture implémentée sur FPGA Virtex-II xcv1000 destinée à des chaînes de traitement en bande de base dans un cadre RLR. Etant destinée à une implantation sur cible FPGA, cette IP à été conçue afin de tirer partie des spécificitées de cette ressource notamment les multiplieurs cablés. De plus, l'accent à été mis sur la reconfigurabilité et la réutilisabilité de l'IP. Ainsi pour l'IP RC-FIR différents paramètres sont accessibles par reconfiguration comme :

- la longueur du filtre (le nombre de cœfficients 8, 16, 32,64),
- le taux de sur-échantillonnage (0,2,4,8),
- la valeur des cœfficients.

La réutilisabilité, quant à elle, est assurée par l'encapsulation de l'IP dans un composant de type GALS. Il est à noter que le filtre reconfigurable (ou RC-FIR) peut être dupliqué et piplené afin de réaliser des filtres plus complexes et performants tels que des filtres 256 ou encore 512 par exemple. La figure suivante présente l'architecture de cette IP.

Sans rentrer dans le détail de cette structure, l'approche de conception retenue ici consiste en la séparation de l'IP en deux parties, une partie contrôle pouvant être reconfigurée et une partie traitement qui est fixe. Les cœfficients du filtre sont modifiables par reconfguration du plan mémoire des ROM et ce de façon partiel ce qui limite le poids du fichier de reconfiguration au minimum. La longueur du filtre ainsi que le taux de sur-échantillonnage sont accessible par paramètres. La flexibilité d'utilisation de cette IP lui permet clairement d'être utilisée dans un contexte RLR. Son intégration au sein d'une architecture HDReM lui permet d'être directement accessible afin de permettre une reconfiguration rapide et efficace.

Réalisation d'un système multiprocesseur reconfigurable dynamiquement

Dans le cadre du réseau d'excellence NEWCOM (*Network of Excellence in Wireless COMmunications*) au département 4 dans le projet D, nous avons mis en place une colla-

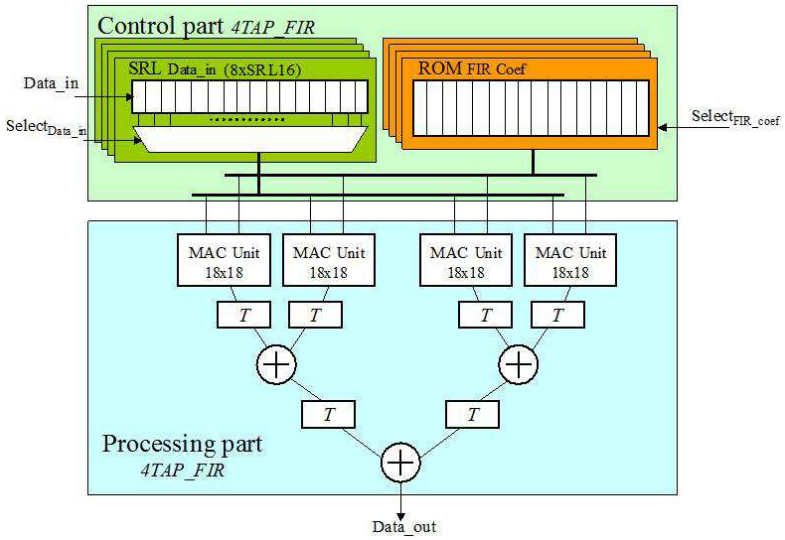


FIG. 1.15: *Filtre FIR reconfigurable dynamiquement*

boration entre SUPELEC, l'UPC de Barcelone en Espagne et RWTH d'Aix la Chapelle en Allemagne. Cette collaboration m'a permis ainsi qu'à un étudiant de l'UPC d'être accueilli en Allemagne, au sein de l'université RWTH, sur une période de deux mois. Le but de cette collaboration était la création d'un système multiprocesseur reconfigurable dynamiquement et de montrer qu'il était possible de fusionner les différentes approches de conception des trois laboratoires afin de réaliser un système pour la RLR. Dans ce projet, chaque participant à apporté son expertise :

- RWTH avec sa maîtrise des ASIP, a mis à disposition sa connaissance des outils de développement LISATek pour la conception du système,
- UPC avec le P-HAL (pour Physical-Hardware Abstraction Layer) a apporté son expertise dans la gestion des données et l'ordonnancement des tâches,
- SUPELEC a apporté son expérience pour la création d'opérateurs communs s'interfaçant avec le P-HAL dans le cadre d'une architecture HDReM.

Le temps accordé pour la réalisation des différents éléments du système ainsi que le temps de prise en main des outils de conception étant assez court, nous avons pris la décision de réaliser un système assez simple. Ceci afin de montrer que les concepts de chacun pouvaient être fusionnés dans la réalisation d'un système et que celui était fonctionnel. Le système retenu, se compose de deux éléments de traitement : un codeur et un entrelaceur. Le système devra pouvoir s'exécuter de deux manière différentes :

- mode : codeur => entrelaceur (codage convolutif classique),
- mode : codeur => entrelaceur => codeur (codage convolutif de type Turbo Code [47]).

Il devra de plus permettre la reconfiguration des cœfficients du codeur en cours de fonctionnement ainsi que l'ordre d'entrelacement. La couche logicielle d'abstraction P-HAL se chargera de l'acheminement des données de traitement et de reconfiguration. Un élément hôte sera créé afin de décider du mode de fonctionnement ainsi que du changement de cœfficient ainsi que l'ordre d'entrelacement. La figure 1.16 illustre la réalisation du système complet.

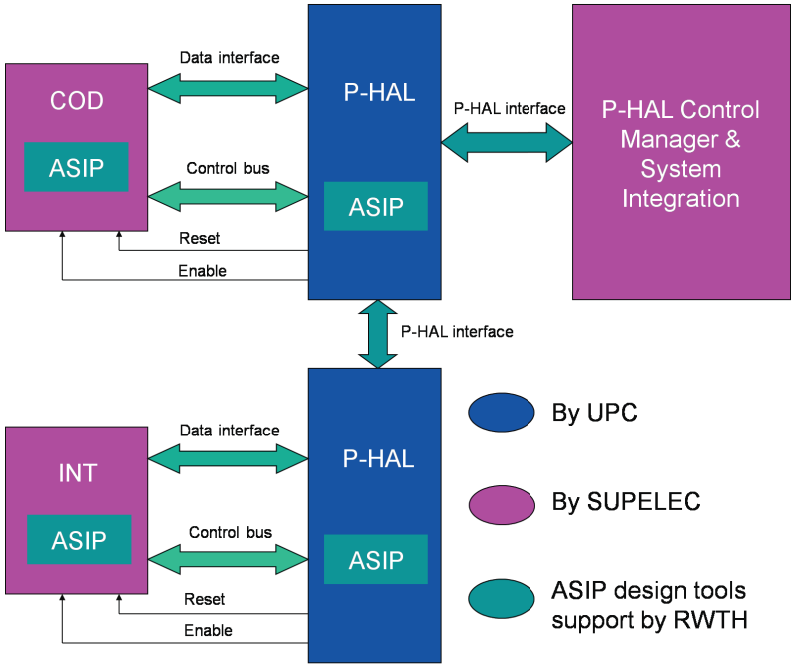


FIG. 1.16: *Système multiprocesseur reconfigurable dynamiquement*

Dans un premier temps nous allons présenter les opérateurs implantés dans le système, puis nous présenterons succintement la couche logicielle d'abstraction P-HAL. Enfin nous parlerons de l'outil de conception LISATek qui nous a permis de développer les éléments de traitement ainsi que la couche logicielle d'abstraction P-HAL sur des ASIP ainsi que la simulation du système complet.

Les éléments de traitement

Les opérateurs retenus pour être intégrés au système sont :

- un codeur : profondeur de 3 et taux de codage un demi,
- un entrelaceur : entrelaçant les données en entrée par paquet de 3 bits.

Comme mentionné précédemment, les opérateurs sont simples mais permettent néanmoins d'éviter des problèmes de conception et de se focaliser sur le fonctionnement glo-

bal du système. Ceux ci sont développés afin d'être dynamiquement paramétrables. Ainsi chacun des opérateurs est contrôlé par un gestionnaire de reconfiguration L3_ReMU ayant pour tâche d'interfacer l'opérateur avec le P-HAL, d'identifier le type de donnée en entrée (donnée de reconfiguration ou donnée de traitement, activation, initialisation, arrêt) et de fournir les indications de fonctionnement de l'opérateur dont il a la charge.

La couche d'abstraction logicielle : le P-HAL

Les premiers travaux sur la couche logicielle intermédiaire qu'est le P-HAL ont été présentés par Revés et al. [14] et sont toujours en cours. L'objectif de cette couche logicielle est d'apporter une abstraction de la couche matérielle en la rendant transparente pour l'application. En plus de doter les composants hétérogène d'une API, le P-HAL permet d'ordonnancer les traitements dévolus aux différents éléments constituants la plateforme radio. La figure 1.17 illustre cette couche d'abstraction.

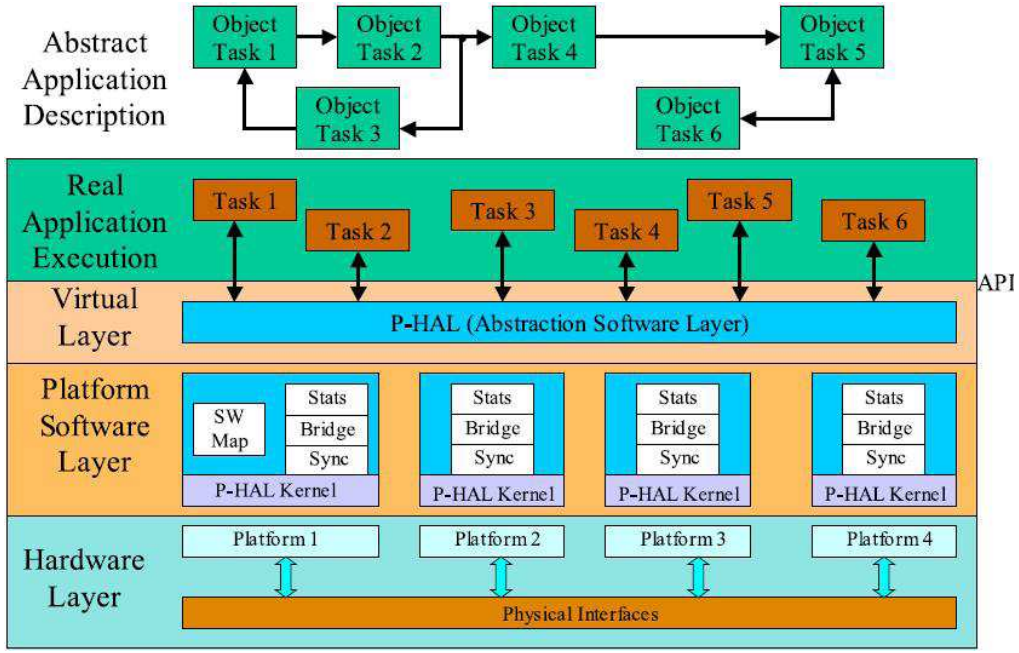


FIG. 1.17: *Couche d'abstraction logicielle P-HAL*

Les principaux mécanismes offert par le P-HAL à travers l'API sont :

- BRIDGE : en charge de l'échange de données entre les objets connectés en temps réel sur un système distribué,
- SYNC : assure la synchronisation des données et des traitements,
- KERNEL : assure l'ordonnancement des tâches et le contrôle des erreurs,
- STATS : renseigne sur le fonctionnement des objets connectés.

À chaque objet connecté au P-HAL est attribuée une adresse permettant de cibler le destinataire d'un message. Le P-HAL est chargé d'ordonnancer le traitement des différentes tâches de traitement. L'architecture de cette couche d'abstraction est organisée comme suit :

- Le Host (ou P-HAL manager) est implanté sur une plateforme Linux et s'occupe de générer les adresses de destination ainsi que les paquets de données (de traitement et de reconfiguration),
- Un premier P-HAL en mode maître est relié au Host et reçoit les paquets de données. Il contrôle l'élément de traitement codeur et s'assure de la destination des données,
- Un second P-HAL en mode esclave est relié au P-HAL maître et reçoit les paquets de données. Il contrôle l'élément de traitement entrelaceur et s'assure de la destination des données.

Conception du système

L'architecture complète du système est illustrée par la figure 1.18.

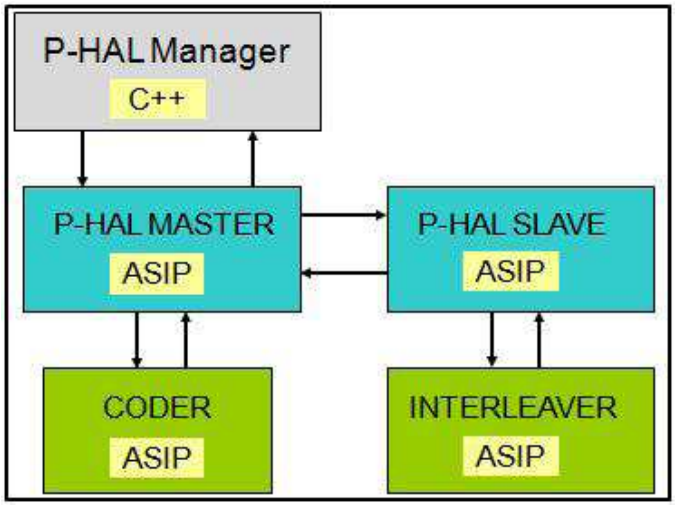


FIG. 1.18: *Système multiprocesseur reconfigurable dynamiquement*

Chaque élément de traitement (codeur et entrelaceur) ainsi que le P-HAL master et le P-HAL slave a été implémenté sur un ASIP possédant un coeur de processeur RISC optimisé et généré par l'intermédiaire de l'outil de conception LISATek. Dans un premier temps les ASIP ont été optimisés pour les besoins du P-HAL mais ils se sont montrés suffisamment généralistes pour permettre le portage des éléments de traitement développés en langage C. Le P-HAL manager quand à lui a été entièrement développé en langage C++.

Résultats

J'ai présenté les résultats de cet échange dans le cadre de la journée NEWCOM Dissemination Day à Paris le 15 février 2007. Le système à été présenté sous la forme de simulation fonctionnelle à travers une interface logicielle utilisateur développée pour l'occasion. La démonstration a été fait de l'ordonnancement des tâches par le P-HAL permettant la réutilisation de l'élément codeur lors d'une exécution en mode codeur => entrelaceur => codeur ainsi que la reconfiguration des divers éléments en cours d'exécution.

Quelques perspectives ont été dressées pour la poursuite de ce projet comme la création d'ASIP spécialisé pour les éléments de traitement ainsi que la réalisation matérielle du système afin de valider physiquement son fonctionnement.

1.4 Conclusion

Après avoir repositionné la problématique de la RL, nous avons effectué un état de l'art du domaine. Puis nous avons présenté l'architecture HDReM issue des travaux de thèse de J.P. Delahaye au sein de l'équipe SCEE. Cette architecture permet une reconfiguration étendue des différents éléments constituant une chaîne de traitement radio. D'une part, la distribution sur trois niveaux d'abstraction permet la prise en compte du contexte hétérogène des plateformes d'exécution. D'autre part, cette distribution permet d'effectuer une reconfiguration multigranularité au sein de l'équipement puisque nous pouvons intervenir au niveau de la reconfiguration entière d'un standard jusqu'à la résolution d'un bogue présent sur l'un des opérateurs d'une chaîne de traitement radio. HDReM répond aux contraintes des architectures RLR en offrant une flexibilité de reconfiguration et une indépendance vis-à-vis des standards passés, en cours ou à venir par une possible mise à jour de ses paramètres opérationnels. Nous disposons donc d'une architecture permettant d'assurer un contrôle précis des ressources de traitement et assurant une flexibilité d'utilisation. Ce type d'architecture permet de répondre aux demandes de reconfiguration mais serait-il possible de lui offrir la possibilité de choisir ses paramètres opérationnels et donc de s'adapter d'elle-même aux variations de son environnement de communication et des besoins des utilisateurs ?

Cette nouvelle approche offrant à un équipement radio les moyens lui permettant de prendre conscience de son environnement ainsi que de ses propres paramètres opérationnels est appelée approche Radio Cognitive et est traitée dans le chapitre suivant.

CHAPITRE

2 La radio cognitive

Sommaire

2.1 La naissance du concept Radio Cognitive

LES sciences cognitives regroupent un ensemble de disciplines scientifiques, comme le montre la figure 2.1, dont le but est l'étude et la compréhension des mécanismes du paradigme du décideur chez l'être humain, l'animal ou l'être artificiel. Dans ce mémoire, nous ferons référence au terme cognitif comme étant un principe de traitement de l'information capable d'acquérir, de conserver et de transmettre des connaissances, de les interpréter et de les utiliser.

2.1.1 Définition et généralités

Introduite par J. Mitola en 1999 dans [2], la radio cognitive est décrite comme l'approche qui permet aux objets communiquants et à leurs réseaux associés d'intégrer l'intelligence nécessaire à la prise de conscience des besoins de l'utilisateur, en termes de

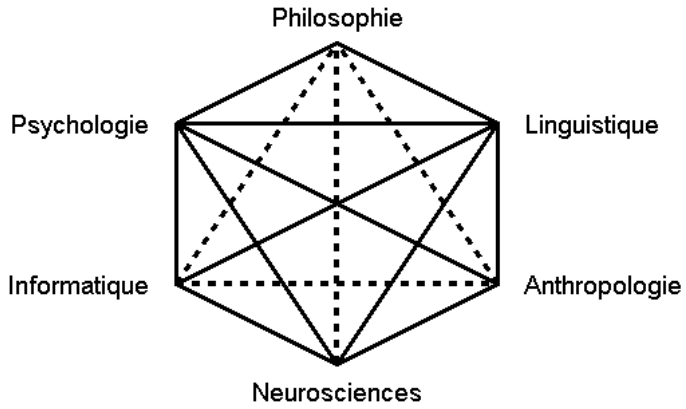


FIG. 2.1: *Hexagramme cognitiviste [48]*

radiocommunication, comme fonction de leur contexte et de fournir les ressources permettant de satisfaire ces besoins. La base même de l'approche radio cognitive est la prise de conscience par l'équipement de son environnement. Cette prise de conscience s'accompagne d'une modification dynamique de ses paramètres opérationnels afin d'offrir à l'utilisateur, à un moment et un endroit donnés, le service désiré avec la meilleure qualité de service (Quality of Service QoS) possible. Un équipement radio basé sur cette approche pourra donc, à l'aide d'un ensemble de capteurs, suivant les besoins des utilisateurs, suivant la charge du réseau et suivant ses propres performances (ressources libres, niveau de batterie, etc.) satisfaire au mieux les besoins de l'utilisateur au moment où celui-ci en a besoin. Cette adaptation devra se faire de manière dynamique et totalement transparente pour l'utilisateur. En effet, ce dernier n'est pas supposé être ingénieur en radiocommunication. Cette intelligence a été modélisée par J. Mitola au travers d'un cycle cognitif illustré par la figure 2.2.

Ce cycle s'appuie sur six points principaux :

- Observe : Prendre conscience de l'environnement par la capture de métriques.
- Orient : Classer les métriques par priorités et aiguiller le traitement (normal, urgent, immédiat).
- Plan : Planifier les meilleures configurations possibles suivant les métriques observées.
- Decide : Allouer les ressources.
- Act : Effectuer la reconfiguration de l'équipement.
- Learn : Apprendre des échecs ou des réussites des précédentes reconfiguration.

On ne peut s'empêcher en regardant ce cycle, de faire une analogie avec la façon dont nous, êtres vivants, appréhendons l'évolution de notre environnement et nous y adaptons. Ainsi à l'aide de capteurs (les cinq sens) nous recueillons un ensemble de métriques qui est traité par notre cerveau. Celui-ci gère les priorités et suivant notre mémoire des faits (phase d'apprentissage) décide d'actions à réaliser pour s'adapter aux

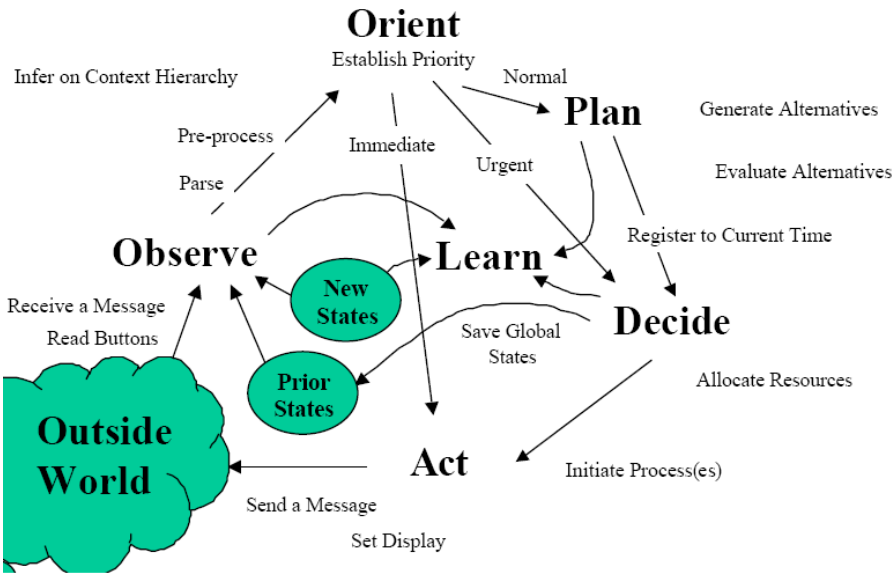


FIG. 2.2: *Cycle cognitif*

changements de l'environnement. Ainsi, soit la situation décrite par les métriques est une situation à laquelle le cerveau a dû faire face et a déjà un plan d'action, soit on se trouve dans une nouvelle situation qu'il va falloir analyser afin de réagir au mieux. Une fois le choix de réaction effectué, le cerveau met en place les moyens permettant l'adaptation et envoie les stimulus aux cibles concernées. Chacun de ces processus est extrêmement complexe et est à l'étude dans le domaine de l'intelligence artificielle. Devenu une quête pour l'humanité, ce besoin de créer une entité intelligente à l'égal des hommes pousse la recherche à perfectionner toujours plus les modèles existants et à faire disparaître de plus en plus les limites entre machines et humains.

On pourra ainsi prendre en exemple le robot *ASIMO* (de Honda) qui est capable de modifier sa trajectoire tout en marchant, de monter ou descendre des escaliers ou encore de courir. Le chien robot *Aibo* commercialisé par SONY qui peut se déplacer, voir son environnement, reconnaître des commandes vocales et évoluer au cours du temps. Dans un autre registre la machine jouant aux échecs *deep blue*, ou encore le moteur *MoGo* développé par l'INRIA jouant au jeu de Go. Les réalisations actuelles en intelligence artificielle peuvent être regroupées en différents domaines tels que :

- Les systèmes experts,
- L'apprentissage automatique,
- Le traitement automatique des langues,
- La reconnaissance des formes, des visages et la vision en général.

L'utilisation de l'ensemble de ces domaines ainsi que leur utilisation au signal radio par un équipement radio communicant lui permet de prendre conscience de son environnement et de lui offrir la capacité de s'adapter à l'évolution de cet environnement.

2.1.2 Contraintes et besoins pour les architectures Radio Cognitives

Il est possible de faire une analogie entre un équipement radio cognitif et un individu (humain ou animal) évoluant en société. Imaginons par exemple un groupe d'oiseaux volant en formation, chacun des membres doit être capable d'évaluer la position de ses voisins ainsi que leur probable futur mouvement en temps réel afin de ne pas les percuter, de même pour un être humain circulant dans une foule. L'analogie se rapporte à un équipement radio devant évoluer au sein d'un ensemble d'équipements dans une même cellule, devant satisfaire au mieux les besoins des utilisateurs sans perturber les autres équipements. Ce type de prise de conscience de son environnement et la manière d'y réagir sont très complexes à modéliser, de nombreux travaux sont réalisés à base d'outils mathématiques comme la théorie des jeux [49], [50] afin d'illustrer et d'analyser les interactions entre équipements cognitifs au sein d'un réseau. Cette discipline mathématique étudie les cas de figure où le sort de chaque participant dépend non seulement de ses propres décisions mais également des décisions prises par les autres participants. Un des objectifs de l'utilisation de cette théorie dans la radio cognitive est de prévenir un cas d'utilisation où un nombre important d'équipements mobiles, au sein d'une même cellule, privilégierait le même ensemble de paramètres de fonctionnement afin de les optimiser pour s'adapter aux variations de l'environnement. Un tel cas de figure révélerait de fortes interférences entre utilisateurs et donc un problème de communication au sein même du réseau et pourrait être illustré par la phrase suivante : le mieux est l'ennemi du bien.

Prenons un exemple, si deux paires de terminaux communiquent sur deux réseaux distincts mais dont les signaux se chevauchent dans le temps et les fréquences, ils vont interférer. À ce moment, les terminaux vont observer ces interférences et les interpréter comme un rapport signal sur bruit et interférence (Signal-to-Interference plus Noise Ratio, SINR) bas. La réaction classique est d'augmenter la puissance de transmission pour augmenter la valeur de ce rapport. Dès lors que l'un des deux terminaux en émission va augmenter la puissance de son signal, l'autre va relever une baisse de son SINR et répond donc en augmentant lui aussi sa puissance. On se retrouve donc enfermé dans un cycle d'optimisation sans fin où les deux terminaux émetteurs vont augmenter leur puissance jusqu'à épuisement des batteries de l'équipement sans pour autant régler le problème. Dès lors soit les deux communications auront un SINR faible et de mauvaises performances soit l'une des deux écrasera complètement la communication de l'autre par sa puissance bien plus élevée. Dans les deux cas ceci représente un échec en termes d'optimisation. Ce cas de figure peut notamment apparaître dans la bande industrielle, scientifique et médicale (ISM) des 2.4 GHz dans laquelle les standards Bluetooth et IEEE 802.11 interfèrent. Bien que dans ce cas le 802.11 possède une plus grande puissance d'émission, le protocole Bluetooth répète l'envoi des paquets jusqu'à ce que la transmission soit un succès ce qui permet la coexistence de ces deux standards mais de manière sous optimale.

Bien entendu le problème d'optimisation soulevé par l'exemple précédent pourrait être évité avec un ensemble de procédés à disposition des équipements radios qui leur permettrait de prendre conscience qu'ils rentrent dans une boucle d'optimisation sans fin et surtout sans résultat. Avec une telle prise de conscience, de tels équipements pourraient modifier leur plan d'adaptation et ainsi intervenir sur d'autres paramètres afin de régler le problème comme modifier la modulation du signal ou le codage canal afin d'améliorer la performance du taux d'erreur de trame (Frame Error Rate, FER) dans le canal. Une autre solution pourrait aussi être la recherche d'une bande libre exempte d'interférences qui amènerait à une reconfiguration des paramètres opérationnels, dont la fréquence porteuse, afin d'y établir une communication.

Au vu des technologies et des connaissances actuelles, il semble improbable d'équiper un équipement mobile d'une telle conscience et d'une telle liberté de contrôle de ses paramètres opérationnels. Ceci pour des problèmes évidents d'intégration au sein de l'équipement, de la consommation des divers algorithmes cognitifs ainsi que de leur complexité de mise en œuvre. Néanmoins, il pourrait être envisageable, dans un premier temps, de restreindre cette conscience et d'appliquer des algorithmes peu exigeants en termes de ressources de calcul et de besoins en mémoire. Cependant, nous pouvons élargir notre vision à un horizon de 10 à 20 ans où les technologies offriront des capacités bien au delà de celle à disposition aujourd'hui. Dans le monde de la RC deux visions différentes sont envisagées. L'une identifiée comme orientée terminal (ou terminal centric) et l'autre orientée réseau (ou network centric). L'approche orientée terminal localise plutôt l'intelligence au niveau de l'équipement, ce qui lui permet d'optimiser ses paramètres opérationnels lui-même afin de rendre optimale la communication. L'approche orientée réseau quand à elle, place plutôt l'intelligence dans le réseau et optimise l'ensemble du réseau pour satisfaire tous les utilisateurs. Ces deux approches étant éloignées en termes de besoins d'optimisation, il en résulte une différence dans les architecture de gestion.

De l'approche orientée terminal (ou Terminal centric) ...

On comprend aisément que l'équipement lui-même est le mieux placé pour connaître son environnement propre. Transmettre ces informations au réseau peut être concidéré comme une perte nette. Ceci est d'autant plus vrai que la mobilité augmente puisqu'il faut mettre à jour ces informations plus souvent. L'idée est donc d'exploiter ces informations dans l'équipement lui-même. Chaque équipement a la possibilité de scruter son environnement et de sélectionner une bande de fréquence sur laquelle il peut transmettre. Il est clair que le choix de la bande reste étroitement contraint par les services utilisables par l'utilisateur (suivant l'abonnement souscrit), les ressources disponibles dans l'équipement (puissance de traitement disponible ou encore niveau de batteries) ainsi que l'environnement même induisant de possibles interférences entre équipements. L'une des méthodes les plus exploitée et intéressante actuellement est l'accès opportuniste au spectre. Cette méthode se base sur la séparation en deux classes des utilisateurs,

l'une nommée *Utilisateur Primaire* (ou UP) et l'autre *Utilisateur secondaire* (ou US). La classe d'UP regroupe les utilisateurs du spectre tels que nous les connaissons actuellement. C'est à dire avec une bande de fréquence allouée et donc fixe attribuée à un service, et souvent moyennant un coût d'exploitation extrêmement élevé comme nous l'avons vu avec les bandes UMTS. La seconde classe d'utilisateurs partage avec les UP leurs bandes de fréquence sous la condition de non interférence. De manière simplifiée, on peut dire que si un US à besoin d'utiliser le spectre radio, il doit s'assurer qu'une bande de fréquence est libre (i.e. non utilisée par un UP comme présenté dans [51]) afin de s'y connecter.

Cette connexion peut intervenir après une prise de décision allant du simple "écouter avant de parler" jusqu'à "penser en fonction de l'environnement". Au moment ou la bande de fréquence occupée de façon opportuniste est réquisitionnée par un UP, l'US à deux possibilités :

- Libérer la bande le plus rapidement possible,
- Rester dans la même bande mais modifier sa puissance d'émission ou son type de modulation afin d'éviter toutes interférences.

Ce type d'accès au spectre n'est autorisé que depuis très récemment. C'est la FCC qui a ouvert la voie en 2003 dans [8]. C'est le premier exemple d'introduction de la RC dans le domaine commercial, mais c'est déjà une réalité, la RC est déjà en marche. La figure 2.3 extraite de [52] schématise un scénario d'accès opportuniste au spectre.

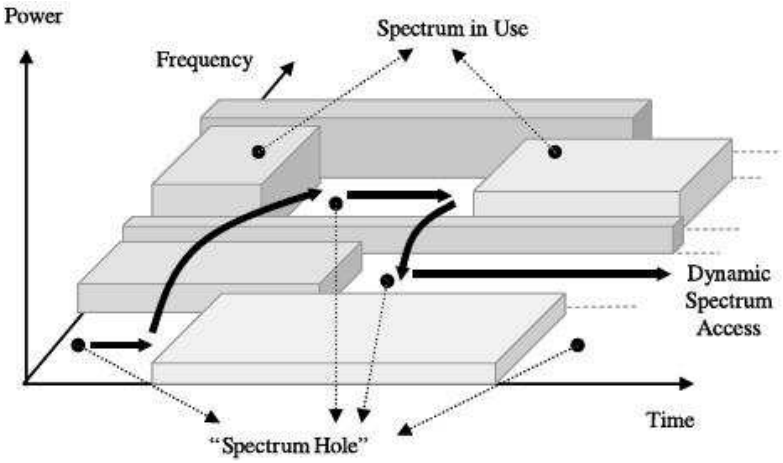


FIG. 2.3: *Accès opportuniste au spectre de manière dynamique*

Un point clé de l'accès opportuniste est que les équipements se doivent d'être à la fois agile dans la gestion du spectre et dans la gestion des règles d'utilisation du spectre. Agile dans la gestion du spectre signifie pouvoir utiliser la gamme la plus large bande de fréquence possible [53], [54]. Un équipement est agile dans la gestion des règles d'utilisation lorsqu'il comprend les règles auxquelles il doit se plier afin d'identifier les fréquences utilisables et comment les utiliser de façon opportuniste. Ces règles pouvant

changer en fonction des UP, de la localisation et de ceux qui gèrent ces règles, un tel équipement doit pouvoir intégrer facilement de nouveaux paramètres et s'y contraindre. Une plateforme nommée *Adaptive Spectrum Radio* (ou ASR) montrant le principe d'accès dynamique au spectre a été décrite dans [55]. Cette plateforme a la possibilité d'adapter sa fréquence et sa modulation afin d'exploiter les trous dans le spectre à la fois dans le temps et dans l'espace des fréquences. Pour cela elle utilise une forme adaptative de multiplexage par division de fréquence orthogonale (ou OFDM pour *Orthogonal Frequency Division Multiplexing*) consistant à répartir le signal sur un grand nombre de sous-porteuses orthogonales modulées individuellement à bas débit. D'autres techniques d'accès au spectre sont à l'étude comme le montre [56] qui présente le standard 802.22 entièrement dédié à la cognitive radio. Ce standard est relié à un groupe de travail nommé *Wireless Regional Area Network* (WRAN) ayant pour but l'utilisation des bandes TV non allouées de façon dynamique. Ce standard porte sur les couches physique (PHY) ainsi que le contrôle d'accès au support (MAC).

Ce que l'on peut reprocher à ce type d'approche est qu'un équipement décidant seul de ses paramètres opérationnels et cherchant à maximiser son efficacité peut entrainer une baisse de QoS pour l'ensemble du réseau. En effet un contexte concurentiel poussé à l'extrême sans aucune entité permettant de réguler les excès peut très vite amener au chaos complet ou seuls les équipements bénéficiant des meilleurs algorithmes pourrait transmettre dans les meilleures conditions. Ceci peut être limité par une approche collaborative pour la répartition des ressources spectrale [57]. Le but de l'approche collaborative est pour un équipement de planifier ses actions en anticipant sur les actions possibles des autres équipements afin d'assurer à l'ensemble un fonctionnement cohérent. Cela implique, et c'est le cas que nous défendons, une coopération entre le terminal et le réseau, et donc une répartition de l'intelligence entre les deux.

. . . à l'approche orientée réseau (ou Network centric) . . .

Cette approche, à l'opposé de la précédente, permet de gérer l'allocation de fréquences par une entité regroupant un ensemble d'équipement radio appelé réseau. Robert Metcalfe, l'un des inventeurs de la norme technique qui donna naissance au réseau internet, énonça un jour une loi : l'utilité d'un réseau est proportionnelle au carré du nombre de ses utilisateurs. Bien que cette notion soit complètement abstraite, prenons le cas d'un réseau ou dix utilisateurs seraient connectés. Suivant la loi de Metcalfe, il serait possible d'évaluer la puissance du réseau à 100, si un onzième utilisateur venait rejoindre ce réseau, alors sa puissance passerait de 100 à 121. Maintenant si l'on prend en compte un réseau regroupant l'ensemble des équipements radio auquel vient s'ajouter de nombreux utilisateurs tous les jours, on imagine les ressources importantes que peut apporter un tel réseau à chaque équipement.

L'approche orientée réseau permet à chaque équipement radio de partager ses informations avec l'ensemble des équipements relié entre eux [54], [58], [59]. Ce partage de connaissance s'effectue via une plateforme centrale qui dispose ainsi d'une connaissance globale de l'environnement et qui applique une allocation optimale des ressources spectrales. Ainsi, chaque équipement transmet un ensemble de variables, relevées dans son environnement proche, à une entité centrale qui se charge de réguler les paramètres de fonctionnement de l'ensemble des équipements suivant les indications relevées. Dans ce cas d'utilisation, l'équipement ne décide pas de ses paramètres opérationnels de communication mais attend un ordre de reconfiguration venant du réseau. La principale difficultée de cette approche est qu'il faut une entité afin de centraliser les données receuilli par les différents équipement. Cette entité va avoir pour rôle de traiter toutes ces données et définir un stratagème d'actions afin de permettre la cohabitation de chaque équipement les uns avec les autres avec une bonne répartition des ressources spectrales. Cela devient d'autant plus complexe que de nouvelles connexions peuvent apparaître de manière totalement fortuite. Les recherches permettant une répartition optimale des ressources sont basées essentiellement sur la théorie des jeux [60], [61], [62].

Ce que l'on peut reprocher à cette approche est d'une part la perte de capacité dûe à la transmission d'informations sur l'environnement et d'autre part, la forte complexité de mise en place du partage de connaissance et de sa distribution ainsi que la complexité du contrôle à mettre en œuvre. De plus un problème inhérent à ce type d'approche centralisée est la latence entre l'arrivée d'un nouvel évènement et sa prise en compte dans la gestion des ressources. Il est possible de diminuer cette latence en utilisant une distribution du contrôle [63]. Ainsi tous les équipements ne sont pas reliés à une seule entité mais sont reliés entre eux par nœuds interconnectés. Chaque nœud peut prendre des décision pour l'ensemble des équipements qui y est relié en accord avec les nœuds voisins.

. . . laquelle choisir ?

Nous l'avons vu précédemment, les deux types d'approche ont chacunes leurs pour et contre. D'ailleurs, une discussion menée lors du premier symposium EU-Japon[1] sur les nouvelles générations de réseaux et le futur d'internet n'a pas permis de départager ces deux approches. Il est clair que ce type de question soulève des problèmes allant au delà du travail de cette thèse. Néanmoins en tant que concepteur d'architectures dédiées aux équipements RC il est essentiel de ne pas trancher à priori pour l'une ou l'autre des solutions et permettre une marge de manœuvre nécessaire afin de s'adapter si l'une ou l'autre venait à l'emporter. Il faut aussi prévoir d'autres types de techniques permettant la répartition des ressources spectrale comme les travaux s'appuyant sur une base de données accessible à chaque équipement renseignant sur les bandes de fréquence à utiliser suivant la localisation de l'équipement [64].

[1] www.ict-fireworks.eu/events/1st-japan-eu-symposium/programme.html

Nous avons vu qu'il existe de nombreuses recherches sur la gestion des ressources spectrales et leur accès par des équipements agiles. Néanmoins il ne faut pas perdre de vue un autre champ de recherche tout aussi important qui est comment fournir à un équipement cette flexibilité de reconfiguration et d'agilité spectrale ? Comment lui permettre de prendre conscience de ces propres paramètres opérationnels et lui apprendre à les modifier suivant l'évolution de son environnement proche ou des indications venant du réseau ? Comment modéliser la connaissance que les équipements peuvent avoir de leur environnement et comment rendre cette connaissance interprétable par un parc hétérogène d'équipements ? Ces questions ont été l'objet de recherches menées dans le cadre de cette thèse et qui ont permis l'établissement d'une architecture de gestion de radio cognitive. Nous allons maintenant présenter les besoins pour la réalisation d'un équipement RC puis nous introduirons la solution que nous avons développée.

2.2 Les besoins d'un équipement radio cognitif

Afin de permettre la mise en place d'équipements RC, nous allons devoir concevoir des systèmes devant être réactifs et conscients afin de s'adapter aux besoins de l'utilisateur. Bien plus complexe qu'une simple prise de décision par dépassement de seuil, l'intelligence fournie à un équipement RC devra lui permettre de :

- Appréhender son environnement,
- Prendre conscience de ses propres paramètres opérationnels,
- Prévoir les changements de son environnement afin de s'y adapter,
- Posséder des facultés d'apprentissage lui permettant d'apprendre de ses réussites et de ses échecs dans le but de toujours réduire le temps nécessaire à une prise de décision menant à une reconfiguration.

Un parallèle peut être d'ailleurs fait entre le comportement humain et les besoins de la radio cognitive [65], d'où l'intérêt dans nos travaux pour les sciences cognitives telles que l'intelligence artificielle, voire les sciences inspirées de la biologie. Il ne faut pas perdre de vue que le contexte radio peut être extrêmement sévère en terme de temps de réaction laisser à l'équipement. Les architectures proposées pour les équipements RC devront donc bénéficier d'une prise de décision rapide et des moyens de mise en œuvre de reconfiguration efficaces.

2.2.1 Les différents moteurs d'intelligence possibles

Cette partie présente de manière succincte différents outils de prise de décision qui peuvent être intégrés à un système RC afin de le rendre cognitif. Bien que l'étude de ces outils dépasse le cadre de cette thèse, il est intéressant de les présenter afin d'appréhender

les mécanismes inhérent à l'avènement d'une prise de décision de reconfiguration au sein d'un équipement RC. En effet, ce travail de thèse consistant à procurer à un équipement RC toute l'architecture de gestion nécessaire au support des fonctionnalités RC, il est primordial d'en comprendre les besoins en terme de prise de décision et d'acquisition de métrique. Ceci dans le but de compléter l'architecture de reconfiguration HDReM.

Les algorithmes génétiques

Ce type d'algorithme se base sur l'utilisation des principes d'évolution et de mutation afin qu'une population atteigne une solution pour un problème donné. Appliqué au domaine logiciel, cette approche permet de résoudre de manière simple et extrêmement efficace des problèmes complexes. Le problème du voyageur de commerce est un cas classique. Le sujet de ce problème est la sélection du chemin le plus court pour un voyageur ayant un certain nombre d'arrêts à effectuer. Ainsi, s'il reste possible de définir de manière énumérative les solutions du problèmes pour un nombre d'arrêts inférieur ou égal à huit, il en est autrement lorsque ce nombre d'arrêts dépasse huit. En effet, les possibilités croissent alors de manière exponentielle.

L'idée de base de cet algorithme est de générer une population de départ de façon aléatoire. Chaque élément ainsi généré se verra attribuer une note correspondant à son degré de confiance vis-à-vis de la solution. Le mécanisme de création de la population de départ doit permettre de générer une population non homogène. Suivant sa performance la convergence vers la solution sera plus ou moins rapide. Puis l'on effectue une sélection des meilleurs individus (ceux ayant le plus haut degré de confiance) sur lesquels on effectue des croisements et des mutations. Les croisements sont utilisés afin de générer de nouveaux individus à partir de deux individus de la génération précédente, la mutation est utilisé afin d'étendre le champ de recherche. Après cette étape on effectue une nouvelle évaluation de la population. Ne connaissant pas a priori la solution, l'algorithme s'arrête quand la solution n'évolue plus pendant un certain nombre d'itérations.

Il apparaît donc que, bien que simple à mettre en œuvre, cette solution peut nécessiter un temps de calcul relativement important (croissance en n.ln(n) ou n est le nombre de variables). Actuellement à l'étude pour une utilisation dans les équipements radio cognitif [66], [67], [68], [69] cette approche doit encore faire face à des problèmes de temps d'exécution et d'occupation mémoire importants en regard des nombreux paramètres environnementaux et opérationnels à prendre en considération pour l'élaboration d'une solution.

Les chaînes de Markov

Cet algorithme est basé sur des processus aléatoires caractérisés par une relation de dépendance des valeurs futures d'un processus aux valeurs présentes uniquement. Ainsi un système est appelé chaîne de Markov s'il possède N états et, qu'à n'importe quel moment le système ne peut se retrouver que dans un seul de ses N états, et que la probabilité qu'il soit dans un état N(t) ne dépend que de son statut au temps (t-1). Le processus de Markov est principalement utilisé dans l'évaluation de la qualité de service ou de performance dans les réseaux de communication mais aussi afin de modéliser les phénomènes d'attente tels que les accès concurrents à des ressources partagés (comme la mémoire, la bande passante ...). Une évolution de cette approche sont les chaînes de Markov cachés (ou HMM pour *Hidden Markovs Models*). Cette approche suppose que le système modélisé est de type markovien de paramètres inconnus. Elle est extrêmement efficace pour les problèmes de reconnaissance de forme ou encore en intelligence artificielle ou en traitement automatique du langage [70]. L'utilisation des HMM se fait en deux étapes, une d'apprentissage et une étape de mise œuvre. L'initialisation se fait par une évaluation initiale des données puis la solution s'affine de façon itérative.

On comprend bien que les HMM peuvent être utilisées dans le contexte RC afin d'effectuer des prises de décision alors que l'environnement ne sera que partiellement connu. Ce type d'approche est par exemple pressenti afin d'estimer la qualité des bandes de fréquences sur laquelle l'équipement va communiquer comme illustré dans [71]. Ce type de modélisation peut aussi être utilisé en entrée d'un moteur d'intelligence comme les algorithmes génétiques pour réaliser la phase d'apprentissage permettant de proposer une ou plusieurs solutions à un jeux de paramètres (qui seraient ici des métriques de l'environnement).

Les systèmes experts

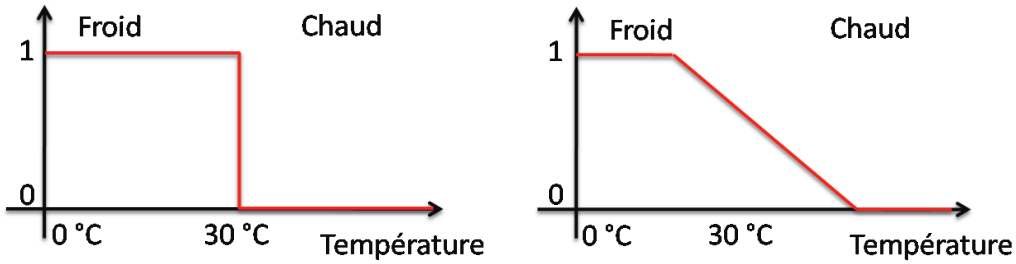
Un système expert, [72], est conçu pour répondre à des questions suivant une base de faits et une base de règles. Il utilise un moteur d'inférence avec ces deux bases en entrées et en déduit la réponse à la question. La plupart de ces systèmes utilisent le raisonnement déductif afin de générer de nouveaux fais à partir des bases à leur disposition. Le type de raisonnement utilisé par le moteur d'inférence est appelé chaînage c'est à dire qu'il va effectuer un raisonnement en appliquant les diverses règles qu'il a à disposition. Différents type de chaînage existe :

- Chaînage avant qui permet de déduire les faits découlant des entrées (approche *data-oriented*),
- Chaînage arrière qui part des faits recherchés et tente de remonter à des faits connus (approche *goal-oriented*),
- Chaînage mixte qui est un mélange des deux précédentes approches.

Un tel algorithme peut être utilisé dans un système RC lorsque les conditions en entrée du modèle sont clairement établies et que les bases de règles ne sont pas trop nombreuses. En effet, si ces bases sont trop nombreuses, il devient difficile de comprendre comment le système expert agit et ainsi valider à coup sur ses décisions. Ce qui oriente le choix de cet algorithme dans un système RC est sa rapidité d'exécution et le compromis satisfaisant entre taille des bases de règles et efficacité de la prise de décision.

La logique floue adaptative

La logique floue (ou *fuzzy logic* en anglais) a été formalisée par Lofti Zadeh en 1965. Cette approche est utilisée aussi bien dans la robotique (reconnaissance de formes), l'automatique (freinage ABS) que l'assurance (sélection et prévention des risques). La logique floue permet de palier aux limites de la logique booléenne en raisonnant à partir de connaissances imparfaites en standardisant la description d'un système complexe par des qualifications linguistiques. Par exemple, la où une description booléenne s'arrête à une description de type tout ou rien, la logique floue permet de rajouter des nuances et de se rapprocher de la réflexion qu'a un être humain. Ce type de différence dans le raisonnement peut être illustré par un problème de consigne régulant la température de l'eau comme illustré par la figure 2.4 suivante :



(a) logique booléenne (b) logique floue

FIG. 2.4: *Différence de modélisation d'un problème*

Ainsi par cet exemple on voit que là où une consigne de type booléenne se contente de représenter le passage du chaud au froid par un simple seuil, l'approche logique floue permet de mieux appréhender la notion de palier dans le réchauffement du liquide.

Néanmoins, l'utilisation de cette approche reste complexe car elle implique l'utilisation de nombreux paramètres corrélés qui doivent pouvoir être exprimés par des critères linguistiques. Une évolution de cette technique est l'utilisation d'algorithmes évolutifs qui sont plus à même de traiter ce genre de problème. La méthode actuellement la plus utilisée dans ce type d'approche est appelé *Fuzzy CoCo* (pour *fuzzy logic and coevolutionary computation*) comme présenté dans [73] ou encore [74].

Les réseaux de neurones

L'approche des réseaux de neurones est un modèle de calcul très fortement inspiré du fonctionnement des neurones présents dans un cerveau. On peut découper le traitement d'un réseau de neurones en deux partie. La première est appelée phase d'apprentissage dont les règles sont présentées dans [75], la seconde partie est la phase de traitement. Un neurone se compose de plusieurs entrées (recevant les données à traiter) auxquelles on applique un poids. Le résultat de toutes les pondérations est ensuite sommé et présenté en sortie du neurone à travers une fonction d'activation réagissant à un seuil. Un neurone présente donc plusieurs entrées pour une sortie comme illustré par la figure 2.5.

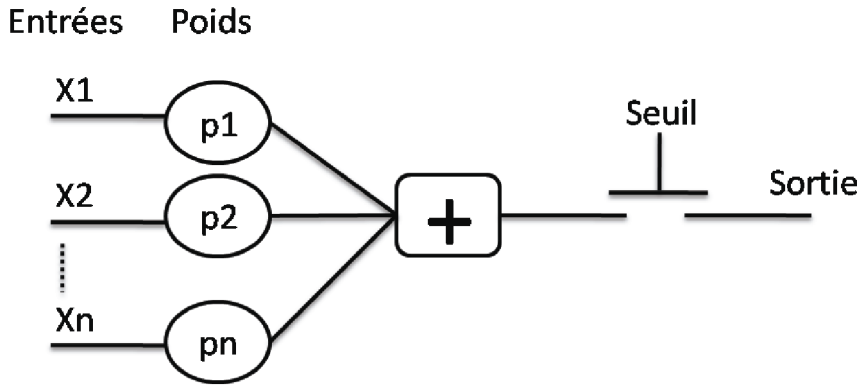


FIG. 2.5: *Constitution d'un neurone artificiel*

La phase critique d'un réseau de neurones est la phase d'apprentissage. Elle se réalise grâce à l'utilisation d'un couple d'entrées/sortie connues à l'avance afin de reproduire de manière informatique les relations entre entrées et sortie. Un cas d'utilisation est par exemple un diagnostique médical qui est en relation directe avec un ensemble de fait. Une fois la phase d'apprentissage finalisée, le réseau de neurones a la possibilité de résoudre des problèmes qu'il n'a pas rencontrés mais qui sont similaires à des cas rencontrés lors de la phase d'apprentissage. On comprend dès lors l'utilité d'une telle approche dans un contexte de radio intelligente comme présenté dans [76], [77] et [3] afin de définir au mieux les paramètres opérationnels de l'équipement suivant un ensemble de conditions ou pour être utilisée comme moteur d'apprentissage.

Nous venons de présenter différents algorithmes pouvant être utilisés afin de rendre un équipement radio intelligent. Ceux ci peuvent être utilisés conjointement afin de renforcer leur efficacité respective. Partons maintenant du principe que nous avons un moteur d'intelligence, de quelle façon peut il avoir connaissance de son environnement ainsi que de ses propres capacités et paramètres opérationnels. Une fois la décision de reconfiguration prise et les nouveaux paramètres choisis, comment gérer de manière efficace cette reconfiguration sur une plateforme hétérogène ?

2.2.2 Les architectures de gestion cognitive

Travailler sur de telles questions, avec comme cible l'intérieur d'un équipement, est à l'heure actuelle assez peu répandu et cela reste encore une activité de niche. De manière générale, les architectures créées afin de répondre aux critères de réalisation d'un équipement RC privilégient une intelligence de type centralisée comme présenté par la figure 2.6.

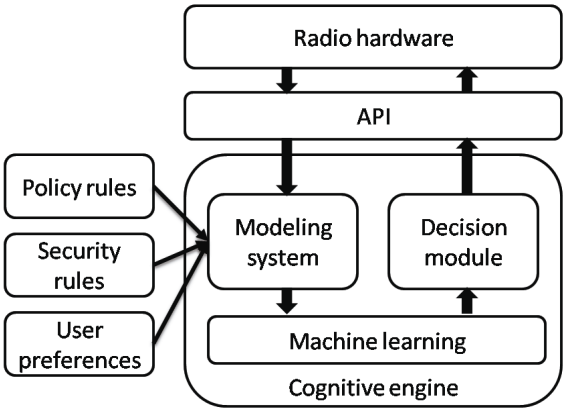


FIG. 2.6: *Intelligence de type centralisée*

On voit ainsi que l'ensemble des paramètres arrivent dans un module de modélisation qui présente alors ces résultats dans une entité permettant de faire des analogies avec différents cas préalablement rencontrés. Ensuite les données sont envoyées au module décisionnel qui décide de la reconfiguration du système et envoie les informations de reconfiguration à la plateforme matérielle. Ce type d'approche est notamment utilisé dans l'architecture *VT cognitive engine* [78], le *BioCR architecture* [79] ou encore l'approche OSCR [80] illustrée ci-après par la figure 2.7.

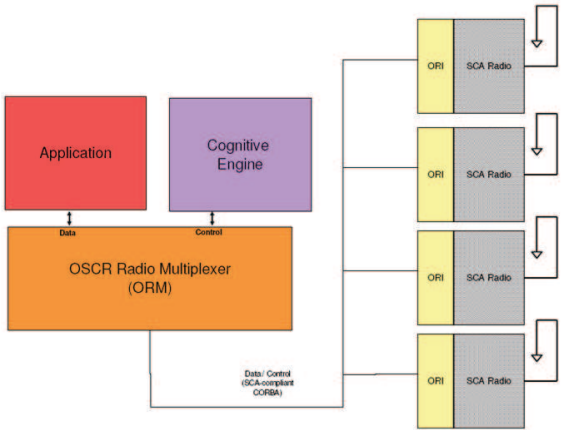


FIG. 2.7: *Architecture OSCR*

L'architecture open source OSCR permet le contrôle d'une ou plusieurs plateformes basées sur l'approche SCA par une entité cognitive.

Bien qu'à l'image d'un cerveau, ces deux approches semblent peu pertinentes pour une utilisation requérrant une réactivité élevée afin de s'adapter au mieux aux variations de l'environnement. En effet, le nombre de données à prendre en compte pour permettre une modélisation globale est extrêmement important. On peut y regrouper de façon non exhaustive :

- les données venant de l'environnement,
- les contraintes utilisateurs,
- les règles d'utilisation du spectre,
- les critères de sécurité en place,
- la modélisation des caractéristiques de la plateforme matérielle.

De plus, l'acquisition, la modélisation, l'interprétation et la prise de décision pour un trop grand nombres de métriques peut consommer un nombre de cycle extrèmement important. Le moteur d'intelligence pourrait alors prendre des décisions non appropriées à un instant donné. Il faut aussi prendre en considération que ce type d'architecture de contrôle de haut niveau pour la partie matérielle ne peut présenter une précision nécessaire à la prise en compte de toutes les spécificités des composants présents sur ce type de plateforme hétérogène. En effet le control des ressources matérielles est permis grâce à l'utilisation d'une couche d'abstraction logicielle de type java. Ce type d'abstraction permet certes une portabilité étendue de l'architecture de gestion (par l'intégration d'interfaces de communication) néanmoins il devient impossible d'accéder aux spécificités des divers composants présents sur la plateforme.

C'est pourquoi, en nous appuyant sur l'architecture HDReM précédemment décrite, nous avons développé une architecture hiérarchique distribuée pour la gestion cognitive au sein d'un système de radiocommunication. Ce travail fait référence à la deuxième ligne de la figure 4, résumé par la figure 2.8, où nous proposons d'associer à chaque élément de l'architecture de gestion de reconfiguration un nouvel élément de gestion cognitive.

Objectives	Means	Results
Description of : **•Functionalities** **•Reality**	**CR equipment specifications** **State of the art**	**HDCRAM**

FIG. 2.8: *Conceptualisation de HDCRAM*

L'architecture ainsi développée dans cette thèse est appelée HDCRAM (Hierarchical and Distributed Cognitive Radio Architecture Management) et est présentée ci-après.

2.3 Une architecture radio cognitive : le HDCRAM

L'architecture HDCRAM illustrée par la figure 2.9 bénéficie de deux parties distinctes ayant chacune un rôle particulier à jouer dans la prise de décision de reconfiguration et la mise en œuvre de la reconfiguration. Ainsi nous complétons l'architecture HDReM présentant les moyens nécessaires au contrôle d'une plateforme hétérogène et pouvant gérer des reconfigurations à multiples granularités par une architecture de gestion cognitive (CRM pour *Cognitive Radio Management*) elle aussi distribuée de façon hiérarchique.

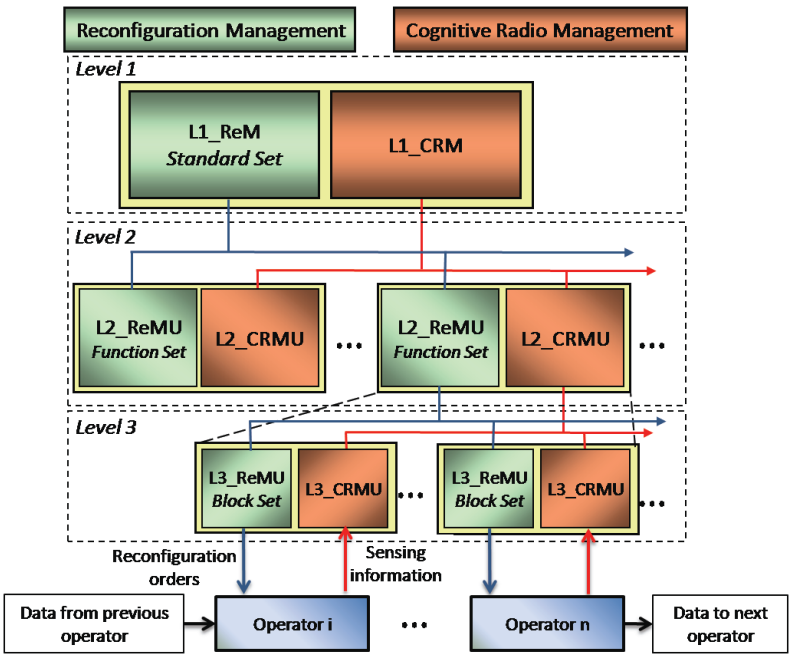


FIG. 2.9: *Interactions entre niveaux ReM et CRM*

À la vue de cette figure, on remarque une surcouche logicielle par rapport à HDReM. Cet ajout permet d'apporter au système de l'autonomie dans la reconfiguration par des prises de décision qui lui sont propres. Le prix à payer par l'ajout de ces agents logiciels reste donc acceptable en vue des bénéfices attendus.

En bas de la figure, nous voyons de quelle manière sont intégrés les opérateurs à HDCRAM. Nous appelons ici de manière générique opérateurs un élément reconfigurable et participant à l'envoi de métriques. Il est bien clair que le choix du rôle d'un opérateur est laissé au concepteur. Ainsi, comme le montre la figure 2.10 différentes possibilités sont offertes suivant les fonctionnalités de l'opérateur.

En 2.10.a, nous voyons un opérateur de traitement classique n'étant pas reconfigurable et ne participant pas à la remontée de métriques. En 2.10.b est illustré un opérateur possédant des facultés de reconfiguration uniquement. En 2.10.c est présenté un opéra-

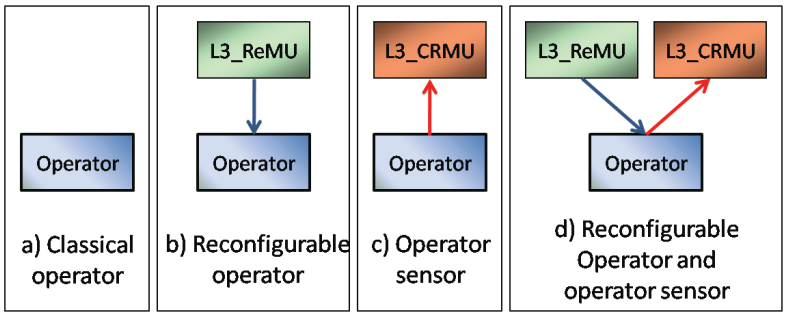


FIG. 2.10: *Possibilités d'intégration d'un opérateur à HDCRAM suivant ses fonctionnalités*

teur participant uniquement à la remontée de métriques (cas d'un capteur de température par exemple). Finalement en 2.10.d nous observons un opérateur présentant des facultés de reconfiguration et participant aussi à la remontée de métriques. Pour ce dernier cas, on peut généraliser le concept de remontée de métriques par le simple fait d'informer sur l'état de l'opérateur (reconfiguration en cours, erreur, etc.). Pour la suite de ce mémoire, par souci de généralité, nous considérons les opérateurs comme étant reconfigurables et participant à la remontée d'informations. Un opérateur sera donc présenté avec un CRMU associé à un ReMU.

Nous allons maintenant décrire la figure 2.9 par la présentation des niveaux de CRM ainsi que leurs relations avec les entités ReM. Dans la partie des ReM, les informations circulent du niveau le plus haut (L1_ReM) vers le niveau le plus bas (L3_ReMU), soit de manière descendante. Dans la partie des CRM, le système va devoir collecter un certain nombre de métriques, les interpréter puis réagir. Il apparaît alors intéressant de faire remonter l'information du niveau le plus bas (L3_CRMU) vers le niveau le plus haut (L1_CRM), comme illustré par la Figure 2.11 publiée en 2006 lors de la première conférence *cognitive radio* Crowncom [81].

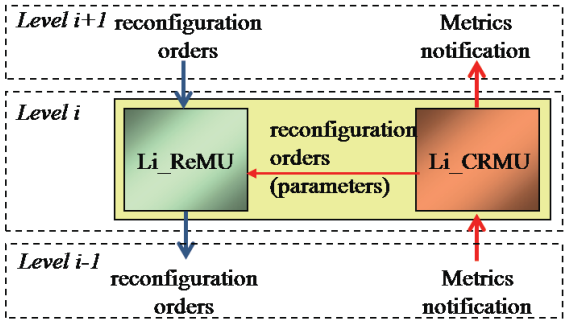


FIG. 2.11: *Interaction entre niveaux ReM et CRM*

En effet mis à part le cas de la demande de reconfiguration effectuée par le réseau ou des demandes des utilisateurs, c'est par l'intermédiaire de métriques collectées par ses opérateurs de traitement du signal ou des données physiques (niveau de batterie, tempé-

rature, etc.) que le système peut s'adapter aux différentes situations. Cette adaptation se fait en contrôlant les ressources de l'équipement afin de fournir la meilleure QoS possible. De plus, ce type de traitement ascendant est parfaitement adapté à la collecte de métriques et à leur classification avant prise de décision. Une des clés de cette architecture est l'union d'une entité de gestion de reconfiguration (ReM) avec une entié de gestion de l'intelligence (CRM) de même niveau d'abstraction. Ce choix permet de séparer le chemin de remontées de métriques à celui de données de reconfiguration. Ce qui est une originalité forte de ce travail puisque nous proposons une séparation fonctionnelle, voire physique si on le souhaite, du chemin du données, du chemin de reconfiguration et du chemin cognitif. Les deux entités ainsi unies communiquent de manière exclusive du CRM vers le ReM afin de transmettre les ordres de reconfiguration relatifs à une prise de décision.

2.3.1 Du rôle de chaque partie

Les entités ReM ont pour fonction d'assurer les reconfigurations des divers éléments de la plateforme hétérogène. Ils doivent être capables d'ordonnancer les différentes tâches de traitement en gérant les priorités. Ils ont accès à des bases de données leur permettant de connaître à tout instant le type de configuration du ou des éléments qu'ils ont a charge ainsi que l'utilisation des ressources matérielles (charge processeur, mémoire disponible, niveau de batterie, répartition des ressources de traitement, etc.). La répartition de ces entités étant hiérarchique et distribuée, il n'est pas utile que chaque entité dispose de l'ensemble des connaissances, ainsi, suivant sa place dans la hiérarchie, une entité ReM aura accès à certaines informations. Par exemple il n'est pas nécessaire à l'entité L1_ReM d'avoir accès au fichier de configuration des ressources matériels (bitstream ou fichier exécutable) mais il est important qu'il ait accès aux bases de données lui permettant de définir le nombre et le rôle des unités L2_ReMU à mettre en œuvre afin d'établir une communication suivant un certain standard.

Les entités CRMU ont pour fonction la capture et l'interprétation des métriques de l'élément dont ils ont la charge (opérateur ou gestionnaire(s) de hiérarchie inférieure). S'ensuit une phase de prise de décision pouvant reposer sur l'expérience de l'entité (expérience acquise par une phase d'apprentissage) ou sur un ensemble de règles préétablies. Une fois prise la décision de reconfiguration de la ressource contrôlée, l'ordre est mis en forme afin de pouvoir être interprété par l'entité ReMU associée. L'efficacité de la reconfiguration est alors vérifiée par la capture de nouvelles métriques. Après que chaque entité CRM (L1_CRM exclu) ait validé la reconfiguration une notification des nouveaux paramètres opérationnels est envoyée au niveau supérieur. Seule l'entité L1_CRM possède une interface avec le monde extérieur et a accès aux bases de données relatives aux politiques d'utilisation du spectre, aux politiques de sécurité, aux instructions du réseau ainsi qu'aux besoins utilisateurs.

Chaque entité de gestion de l'intelligence CRM(U) dispose de connaissances et de comportements privés ainsi qu'une capacité d'exécution propre et participe à l'adaptation dynamique de l'ensemble de l'équipement radio. On voit donc une analogie entre les entités CRM(U) et les agents logiciels [82] définis dans les sciences informatiques. En effet, chacune de ces entités de gestion ne dispose que d'une représentation partielle de son environnement, ainsi un CRM(U) n'aura aucune connaissance des objectifs et des ressources à disposition d'une autre entité de gestion de l'intelligence d'un même niveau. Un CRM(U) a comme comportement la satisfaction de ses objectifs en tenant compte des ressources et des libertés à disposition. Pour ce faire chaque CRM(U) à un comportement autonome dans le sens ou il est capable de prendre des initiative et grâce à sa flexibilité, peut gérer ses propres ressources à disposition et générer de nouveaux paramètres de reconfiguration. De plus, chaque CRM(U) peut bénéficier d'algorithmes lui permettant de s'adapter dans le temps et d'apprendre de ses succès comme de ses erreurs afin d'améliorer le traitement de ses objectifs.

L'un des points noirs des architectures conçues à base d'agents logiciels est le fait que chaque entité n'ayant qu'une connaissance limitée de son environnement, il est difficile de faire converger une solution vers un but commun. Nous pallions ce type de problème par un modèle de contrôle en trois couche permettant de réguler le traitement des agents par une unité ayant une représentation plus globale de son environnement. Une couche prend le contrôle lorsque la couche antérieure ne peut plus contribuer, par ses opérations, à l'accomplissement des buts. Ainsi une entité de niveau hiérarchique supérieur a la possibilité de contrôler comme défini dans [83] (par l'intermédiaire des libertés laissées aux agents logiciels contrôlés), le travail des unités CRMU qui lui sont associées afin de créer :

- une coopération (les agents sont libres et peuvent fonctionner de manière concurrentielle)
- une coordination (limiter les interactions néfastes et exploiter celles qui sont bénéfiques),
- une négociation (limiter certains agents afin de laisser une plus grande marge de manœuvre à d'autres).

Le flot de donnés dans la partie CRM étant ascendant, c'est par l'intermédiaire de son entité ReM associée que ce contrôle est effectué. En effet, les entités ReM et CRM associées partageant la même base de données, il est possible pour un ReM de venir contrarier ou libérer les degrés de liberté de son CRM.

2.3.2 De l'acquisition des données à la réaction du système

Nous allons présenter ici les moyens supportés par notre architecture afin de permettre une prise de décision de reconfiguration suite à une acquisition de métrique. Cette prise de décision découle d'un certain nombre d'actions présentées par la figure 2.12.

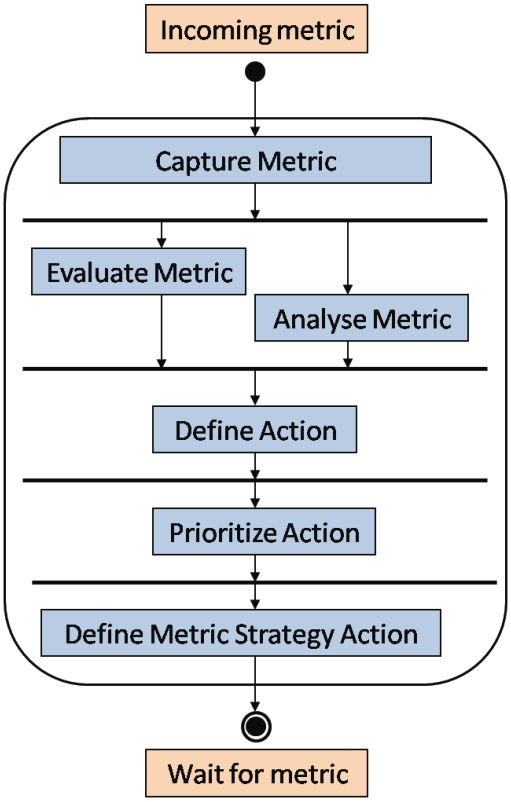


FIG. 2.12: *Cycle cognitif*

La figure 2.12 illustre l'ordonnancement des tâches du point d'entrée qui est l'acquisition de la métrique, à la sortie qui est la mise en œuvre ou non d'une reconfiguration. On voit ainsi qu'il existe cinq stades avant qu'un gestionnaire cognitif ne sorte une information pouvant mener à une reconfiguration.

Les opérations sont définies comme suit :

- Evaluate metric : évalue l'importance la métrique,
- Analyze metric : identifie l'origine de la métrique,

Le résultat de ces deux opérations est indiqué par une valeur (par exemple : basse, moyenne, haute).

- Define action : défini par le concepteur au départ mais pourrait profiter de l'auto apprentissage,

- Prioritize : définit la priorité suivant l'impact de la métrique, la plus haute étant accordée à l'action entraînant l'impact le plus important sur le système,
- Define Metric Strategy Action : l'action prise ici peut être une simple transmission de la métrique (dans un format approprié) au niveau supérieur ou la transmission d'un ordre de reconfiguration au ReM(U) dont il a le contrôle puis l'envoi de la modification effectuée au niveau supérieur de gestion de l'intelligence.

L'un des points clés de l'architecture HDCRAM est que chaque entité CRM dispose de son cercle cognitif afin d'offrir la possibilité de boucler au plus vite le cercle cognitif pour une réaction immédiate comme présenté par la figure 2.13.

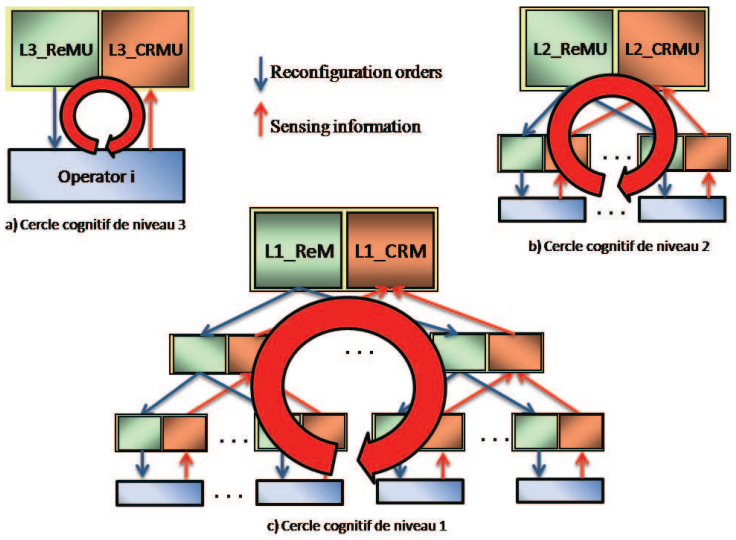


FIG. 2.13: *Différence de rapidité de traitement*

Cette liberté permet par exemple de résoudre au plus vite un bogue d'un élément de traitement par un rechargement immédiat du fichier de reconfiguration ou encore la gestion d'un conflit de ressource. Il est important de comprendre que bien que le cycle garde la même nomination d'un niveau à l'autre, les degrés de liberté offerts ainsi que les algorithmes de décision devant être mis en place sont de natures différentes suivant la granularité du niveau. L'utilisation de ce cercle cognitif au sein de chaque entité CRM permet de réduire les temps de réaction du système et d'assurer une forte flexibilité au niveau de la reconfiguration.

Il est à noter que les algorithmes de prise de décision au niveau L3 seront probablement moins complexes donc moins exigeants en ressource de calcul que ceux utilisés au niveau L1. Ceci s'explique simplement par le nombre de métriques à traiter par l'entité de gestion de l'intelligence ainsi que les degrés de liberté offerts à ce bloc sur la reconfiguration d'un ou plusieurs éléments de la chaine de traitement radio.

Cette différence de complexité au niveau de la prise de décision peut être exprimée par différents moyens de traitement, on peut par exemple en fournir une liste non exhaustive (par ordre croissant de complexité) :

- niveau de seuil,
- machine d'état,
- réseau de neurones,
- heuristiques, algorithme génétiques,
- chaînes de Markov Cachées,
- apprentissage par renforcement,
- ou encore une combinaison de tout cela.

2.3.3 Détail du fonctionnement de l'architecture

Nous allons maintenant décrire le rôle des entités CRM suivant leurs niveaux de hiérarchie ainsi que leurs interactions avec leurs entités ReM associées. Cette architecture à pour objectif de pouvoir répondre à n'importe quel contexte qui se présentera dans un équipement RC. On peut ainsi penser à l'intégration dans HDCRAM de capteurs non reconfigurable mais participant à la prise de décision par l'acquisition de métrique, les opérateurs de traitement du signal reconfigurable n'ayant pas de but RC. À noter que pour cette dernière catégorie, l'ajout d'un gestionnaire d'intelligence peut permettre la validation de la reconfiguration par exemple.

La figure 2.14 montre de manière non exhaustive une unité L3_CRMU et ses dépendances.

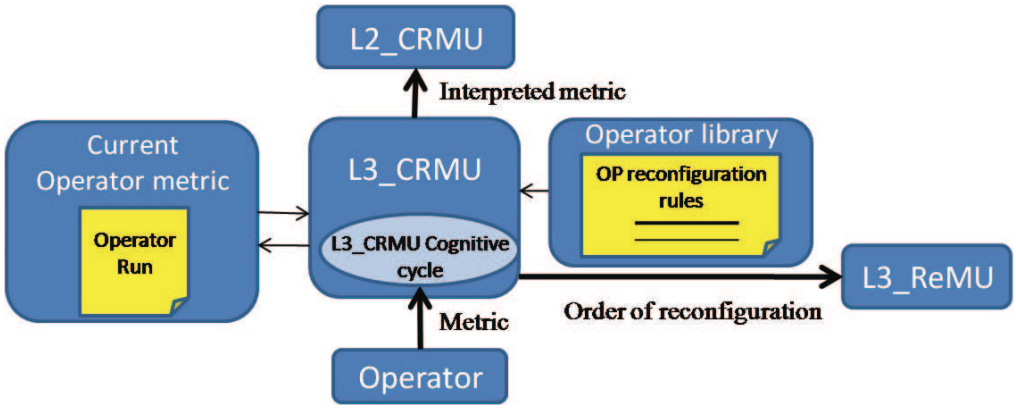


FIG. 2.14: *Niveau de gestion de l'intelligence L3_CRMU*

De même que lors de la présentation de l'architecture HDReM, on peut observer sur la gauche de la figure ci dessus la partie de la bibliothèque (partagée avec l'unité ReMU associée) où sont stockées les données relative au fonctionnement courant de l'opérateur.

A droite est présentée une autre partie de la bibliothèque où sont stockés des données relatives à l'entité contrôlée. Dans le cas présent, l'entité contrôlée étant un opérateur, cette partie de la bibliothèque peut contenir des routines de reconfiguration ou encore un certain nombre de paramètres de fonctionnement.

En fonctionnement, un L3_CRMU collecte une ou plusieurs métriques à partir de l'opérateur dont il a la charge. Après avoir effectué une acquisition de métriques, cette unité effectue une interprétation et réagit. La réaction peut être d'adresser un ordre de reconfiguration à son L3_ReMU associé entraînant alors la reconfiguration de l'opérateur ou, si cela ne s'avère pas nécessaire, transmettre le résultat de l'interprétation à son L2_CRMU associé.

Nous avons donc ici un premier niveau d'abstraction entre les métriques envoyées par l'opérateur à son L3_CRMU, qui reflète une valeur physique, l'interprétation qui en est faite et la finalité qui peut être :

- une simple transmission au L2_CRMU associé sans prise de décision de reconfiguration,
- une transformation en ordre de reconfiguration adressé au L3_ReMU associé.

En effet l'interprétation exprime le comportement de l'opérateur et permet d'évaluer sa performance. Un exemple serait l'estimation de la performance d'un opérateur (valeur de TEB, SIR, SINR, etc.) par des valeurs telles que bas, moyen ou haut. Il est donc possible d'utiliser pour ce faire des algorithmes de logique floue permettant ce genre d'abstraction linguistique.

Un L2_CRMU reçoit donc un certain nombre d'informations interprétées par l'ensemble des L3_CRMU dont il a la charge. Cet ensemble de métriques va permettre une évaluation globale au niveau de la fonction cette fois. On rappel que la notion de fonction est ici utilisée pour désigner un ensemble d'opérateur. Un L2_CRMU a donc pour rôle d'ajuster les performances des L3_CRMU, dont il a la charge, de manière à s'acquitter des performances fixées par le L1_ReM pour la fonction qu'il doit réaliser suivant les ressources disponibles.

De la même manière que pour le niveau précédent, le niveau de gestion de l'intelligence L2_CRMU peut ordonner une reconfiguration par l'intermédiaire de son L2_ReMU associé. Ce qui se traduira par la reconfiguration d'un ou plusieurs L3_ReMU à charge. Ces derniers effectueront alors une reconfiguration de l'opérateur dont ils ont la charge. À ce niveau on pressent déjà une notion de compromis entre la performance globale à atteindre au niveau de la fonction à réaliser et des performances des opérateurs qui la composent. En effet, il est probable que les opérateurs ne doivent pas être à tous prix paramétrés dans les meilleures configurations pour obtenir la performance fixée par le L1_ReM pour la fonction. La figure 2.16 montre le niveau cognitif L1_CRM et ses dépendances au sein de l'architecture hiérarchique distribuée.

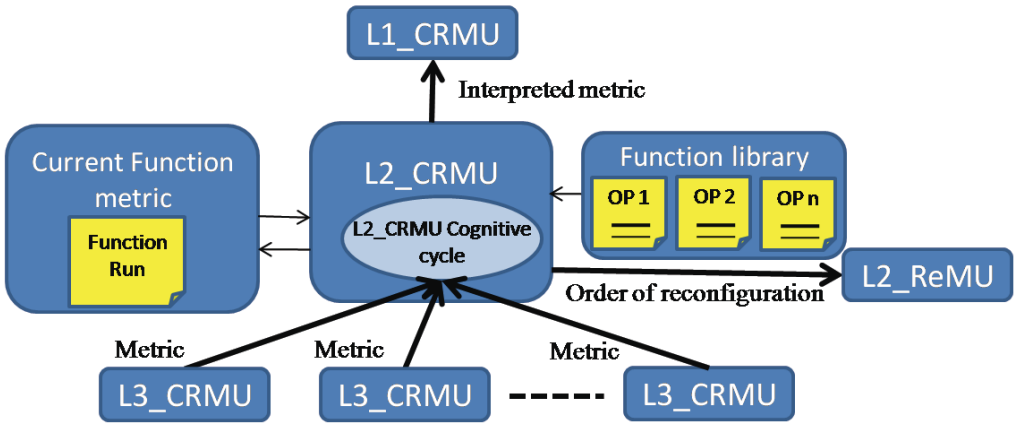


FIG. 2.15: *Niveau de gestion de l'intelligence L2_CRMU*

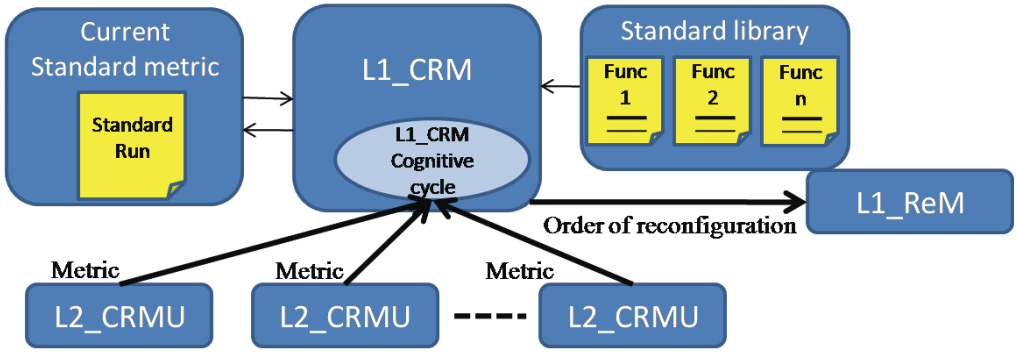


FIG. 2.16: *Niveau de gestion de l'intelligence L1_CRM*

Les métriques, relevées et interprétées par les différents L2_CRMU, sont ici analysées et interprétées. Cette interprétation peut conduire à une paramétrisation d'une ou plusieurs fonctions par l'intermédiaire du L1_ReM. Une fois encore l'optimisation d'une fonction pour la réalisation du standard de communication peut être effectuée par un compromis sur les performances des fonctions utilisées afin de satisfaire à une QoS demandée par l'utilisateur. Ce cas pourrait par exemple survenir lors d'un relevé du niveau de la batterie indiquant un niveau relativement bas. L'objectif du système étant de permettre l'établissement d'une communication, il doit effectuer un compromis entre consommation et performance suivant les degrés de liberté dont il dispose.

2.3.4 Exemples d'application

Encapsulation d'un opérateur de traitement

L'architecture HDCRAM peut tirer parti de l'approche par "opérateur commun" pour offrir une flexibilité accrue au niveau de la reconfiguration ainsi qu'un maximum de réutilisation des différents blocs de traitement. Prenons le cas d'un opérateur conçu pour une utilisation sur une architecture HDReM. Afin de satisfaire le concept "opérateur commun", l'opérateur se doit d'offrir d'une part une flexibilité de configuration au moment de la conception de manière à être facilement porté sur différentes plateformes. Cela est appelé "*reusability at design time*". D'autre part, une reconfiguration à l'exécution (ou "*reusability at run time*") est aussi importante afin de s'assurer que seules les ressources nécessaires au changement de contexte sont reconfigurées, et permettre ainsi un changement de configuration le plus rapide possible. La figure 2.17 illustre l'encapsulation d'un composant de traitement dans un module de type GALS.

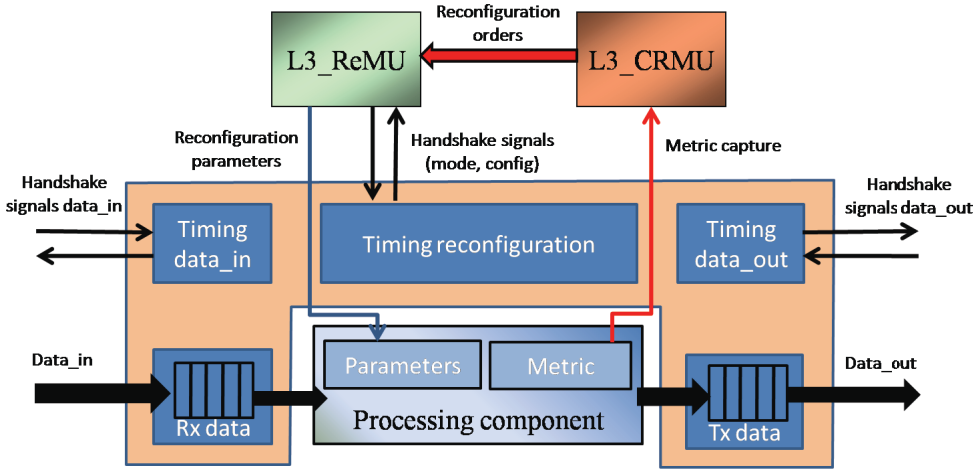


FIG. 2.17: *Encapsulation de type GALS*

La partie contrôle de cet opérateur est assurée par un L3_ReMU développé sous forme de machine d'état (par exemple) permettant, par changement de paramètres, de modifier le comportement de l'opérateur. Le L3_ReMU est contrôlé par un L3_CRMU qui peut, suivant les besoins, effectuer des passages de paramètres au L3_ReMU associé pour reconfigurer l'opérateur. Cette demande de reconfiguration est accompagnée d'une priorité de reconfiguration laissant ainsi la gestion de la reconfiguration à l'entité L3_ReMU. Ainsi, en cours de fonctionnement, l'opérateur fournit à un intervale de temps donné (à définir suivant le composant) une ou plusieurs métriques à son L3_CRMU qui les analyses et décide ou non d'une reconfiguration. Il est à noter que lorsqu'une reconfiguration est effectuée, la validation de cette reconfiguration est effectuée par l'unité L3_CRMU.

Matrice d'interconnexion reconfigurable

Nous pouvons reprendre l'exemple du re-routage de données illustré dans le chapitre 1 avec l'ajout d'une entité de gestion L2_CRMU comme présenté dans la figure 2.18. La présence du gestionnaire d'intelligence L2_CRMU donne la possibilité d'effectuer une reconfiguration contrôlée du chemin de donnés sans ordre venant du gestionnaire de reconfiguration de niveau supérieur, le L1_ReM.

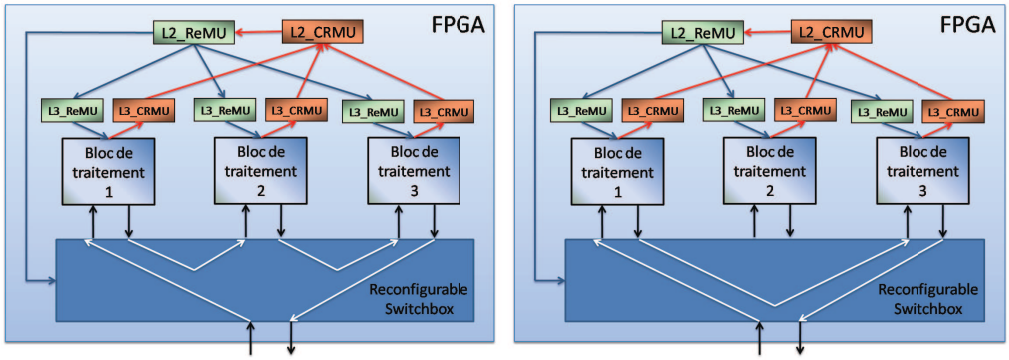


FIG. 2.18: *Modification du chemin de donnée après décision du L2_CRMU*

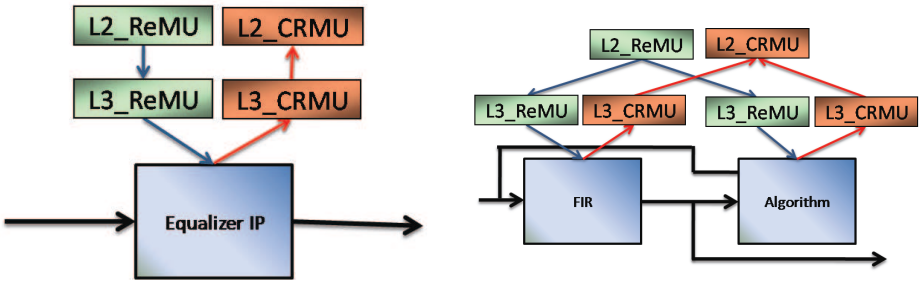
Ce genre de contrôle trouve son utilité dans des applications ou la présence d'un élément de traitement dans la chaîne radio dépend de certains paramètres mais aussi lorsqu'apparaît la nécessité de réallouer un opérateur de traitement sur une nouvelle cible d'exécution. Bien entendu, lorsqu'un évènement tel que la réallocation d'un opérateur sur une nouvelle cible d'exécution survient, le L2_ReMU doit ternir compte du temps d'acheminement des données entre ces divers opérateurs dans son ordonnancement des tâches. Cette faculté de gestion d'éléments implantés sur différentes cibles matérielles est possible grâce à l'utilisation d'une table de routage qui maintient l'adressage logique entre les éléments sous le contrôle du couple L2_ReMU/L2_CRMU. Il est évident que la matrice d'interconnexion ne participe pas à l'envoi de métriques, ainsi, lorsqu'un ordre de reconfiguration est généré à destination du L2_ReMU par un L2_CRMU, il ne contient pas d'informations spécifiques sur les moyens de mise en œuvre. C'est le rôle du L2_ReMU d'interpréter cet ordre et d'effectuer les reconfigurations nécessaires sur la matrice d'interconnexion.

Flexibilité d'utilisation

Nous avons tenu à ce que l'utilisation de notre architecture soit la plus aisée possible et fournisse une flexibilité de mise en œuvre importante. Ainsi, il est acceptable de considérer que si un opérateur n'est pas reconfigurable et ne participe qu'à la capture de métrique, il est possible de ne lui associer qu'un gestionnaire d'intelligence CRMU

qui effectuera alors une simple interprétation puis remontée de métrique au niveau suprieur de hiérarchie. De même si l'opérateur ne fournit aucune métrique directement exploitable, il est possible de lui adjoindre uniquement un gestionnaire de reconfiguration ReMU. Il est aussi possible dans un premier temps de n'associer que quelques opérateurs aux gestionnaires afin de permettre un passage en douceur vers une architecture HDCRAM complète. Ceci est primordial pour éviter un refus total par l'industrie de la téléphonie de l'introduction de telles fonctionnalités.

Le concepteur aura de plus une totale liberté quand au choix de la granularité d'un élément de traitement comme le montre la figure 2.19.



(a) stucture d'égaliseur monobloc (b) stucture éclatée d'égaliseur

FIG. 2.19: *Différence de structure pour réaliser un égaliseur*

En effet, du point de vue du couple L2_CRMU/L2_ReMU (et donc au niveau fonctionnel) l'objectif est le même, contrôler un égaliseur. La différence se fait sur l'étendue des possibilités de reconfiguration de l'opérateur. Dans le cas 2.19(a) nous avons l'utilisation d'un bloc d'IP conçu dans une optique de performance pure dans le sens ou elle est optimisée pour un champ réduit d'égalisation. L'approche utilisée dans 2.19(b) montre quant à elle l'utilisation de deux éléments pour la réalisation de l'égalisation. Il y a ainsi un élément de traitement de filtrage (le FIR) et un élément réalisant un algorithme permettant de le contrôler. Ce type de structure permet une granularité de reconfiguration assez étendue puisqu'il est possible de réaliser tout type de FIR, en pipelinant plusieurs éléments FIR (exemple du FIR illustré au chapitre 1) ou en lui administrant de nouveaux paramètres de fonctionnement, mais aussi tous types d'algorithmes. De plus, si la cible d'exécution est un FPGA de la famille Xilinx, il est possible de reconfigurer l'un des deux éléments sans perturber le fonctionnement du deuxième. Il est clair que cette flexibilité se paye par une occupation de surface plus importante dans le cas 2.19(b) ainsi que l'ajout d'un couple CRMU/ReMU. Ce type de compromis entre performance/flexibilité est laissé à l'entière disposition du concepteur qui pourra compter sur le support de notre architecture quel que soit le choix retenu.

Contrôle du Front-end

L'utilisation de notre architecture ne se limite pas à un contrôle des éléments de traitement numérique mais peut aussi être utilisé pour le contrôl des éléments du Front-end analogique comme les antennes, les filtres ou encore les convertisseurs. En effet, de récents travaux permettent par exemple de modifier la largeur de bande ainsi que la dynamique des convertisseurs [84], grâce aux technologies MEMS (*Micro Electro Mechanical Systems*) qui permettent de modifier également les bandes de travail des filtres [85]. De même il est aussi possible de contrôler les antennes afin de modifier leurs directivitées en émission ou en réception en gérant la phase sur un réseau d'antennes comme présenté par la figure 2.20.

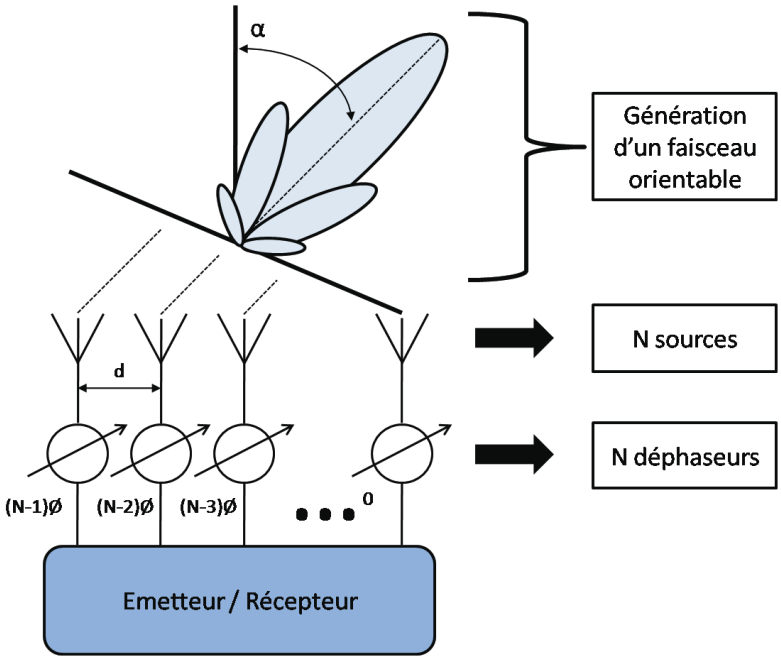


FIG. 2.20: *Antenne adaptative*

De plus, ce type d'antenne peut être conçue pour une utilisation multi-bandes. On imagine bien l'apport de ce type de technologie dans un contexte RC. En effet, le fait de posséder ce type d'antenne permet à un équipement RC d'explorer l'ensemble des fréquences du spectre ou encore d'améliorer la qualité de réception ou d'émission par une meilleure focalisation du faisceau. Il est possible d'aller encore plus loin en utilisant des antennes dite adaptatives. Ce type d'antenne permet de découper l'espace de rayonnement en micro espace contenant chacun un diagramme de rayonnement maximisant la qualité d'émission/réception tel que présenté par la figure 2.21 extrait de [86].

De nombreux projets ont été à l'étude concernant ces antennes agiles, on peut nommer le réseau d'excellence AMICOM [87] qui s'est terminé en novembre 2007 suite à trois années de recherche et ayant reçu la mention "terminé avec succès" de la part de la

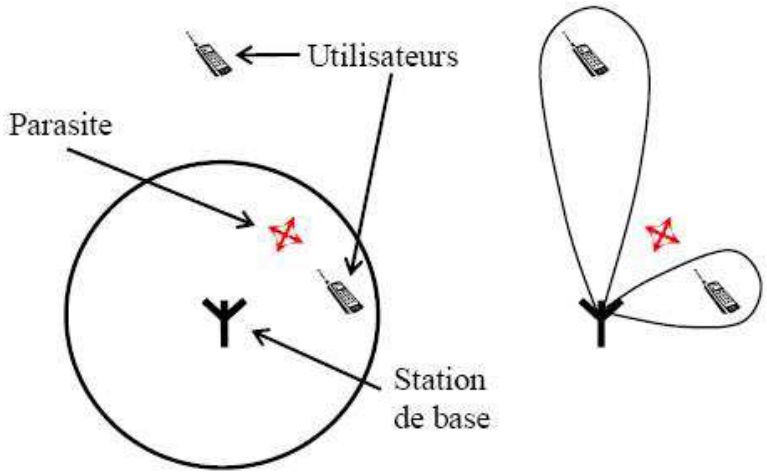


FIG. 2.21: *Adaptation du diagramme de rayonnement de façon dynamique*

commission européenne de même que le réseau d'excellence ACE [88] lui aussi achevé. On peut aussi nommer le projet ARROW [89] qui a pour but de dresser la feuille de route de la technologie MEMS (et NEMS ou *Nano Electro Mechanical Systems*) à travers leur évolution technologique ainsi que d'évaluer leurs possibles applications.

Les différents éléments agiles du front-end que nous venons succintement de présenter peuvent être facilement gérés par notre architecture. En effet, nous avons précédemment mis en évidence le fait que le concepteur avait la liberté d'utiliser pour le contrôle d'un opérateur soit un couple l3_CRMU/L3_ReMu soit une seule unité L3_CRMU soit une seule unité L3_ReMU suivant ses fonctionnalités. Dans ce cadre d'application, on étend la notion d'élément de traitement aux éléments participant à la réception ou la création de la forme d'onde. Si l'on prend le cas d'une antenne, en cas d'utilisation d'une antenne large bande classique, il ne sera pas possible d'effectuer une reconfiguration dynamique de l'élément. Il n'est donc nullement utile de créer une unité de gestion L3_ReMU par contre, les métriques relevées par l'antenne peuvent être interprétée par un L3_CRMU et participer à l'adaptation du système à l'environnement. Si l'antenne utilisée possède des dispositions à la reconfiguration, alors une unité L3_ReMU pourra être conçue afin de permettre le contrôle de cette reconfiguration. Notre HDCRAM peut ainsi s'adapter aux possibilités du matériel.

2.4 Conclusion

Nous avons présenté dans ce chapitre une architecture permettant de gérer de manière rapide et efficace des plateformes de type hétérogène par l'intermédiaire de son gestionnaire hiérarchique distribué de reconfiguration. De plus cette architecture permet au système d'appréhender son environnement et de s'y adapter de manière rapide et flexible

grace à un gestionnaire hiérarchique et distribué de l'intelligence. En tout début de chapitre nous avons vu que se posait la question du siège de l'intelligence : plutôt locale à l'équipement ou plutôt déportée dans le réseau ? Notre architecture peut être utilisée quelque soit la solution retenue, la différence se fera dans la complexité des algorithmes d'intelligence présens dans les CRM(U).

Le domaine de la radio cognitive étant pluridisciplinaire et requerrant par là même l'utilisation de nombreux outils de dévelopement différents, il est extrêment difficile de fournir une architecture réutilisable par la communauté du domaine, ce qui en limite fortement l'intérêt. Notre but est de développer non pas une solution spécifique et optimisée à un problème mais une solution flexible et générique devant s'adapter à tous les scénarios RC possibles, et pouvant être portée sur un parc matériel étendu. L'arrivée des sciences informatiques, en raison de la part de plus en plus importante du logiciel au sein des plateformes, apporte de nouvelles méthodes de travail aux experts matériels. En effet, les problèmes de compatibilité inter-logiciels ont déjà été abordés dans les sciences informatiques et résolu par l'adoption de langages de haut niveau permettant de passer outre les problèmes de la cible d'exécution.

Afin d'offrir une flexibilité et une réutilisabilité maximale pour notre architecture HDCRAM, nous avons donc retenu un langage unifié de programmation de haut niveau permettant de s'abstraire des contraintes matérielles. Ce langage et la modélisation de notre architecture sont présentés dans le chapitre trois.

CHAPITRE

3 Un métamodèle exécutable

Sommaire

3.1 Une nouvelle approche de conception

NOUS avons vu précédemment qu'un équipement RC reposait sur l'utilisation de plateformes hétérogènes nécessitant différents outils de conception n'offrant pas d'inter-opérabilité. De plus le parc d'équipements étant lui même hétérogène par nature (renouvellement technologique) il est difficile de permettre la réutilisation de blocs de traitement d'une série d'équipement à une autre (ceux ci étant fortement liés à la cible d'exécution). L'arrivée de logiciel pour le contrôle du matériel dans les équipements radio va fortement modifier cet état de fait.

En effet, l'arrivée des sciences informatiques apporte de nouvelles méthodes de conception. On peut ainsi nommer l'approche MDE (Model Driven Engineering) dont le but est

de modéliser en début de cycle les fonctionnalités d'une application sans se soucier des exigences telles que l'efficacité ou les contraintes matérielles par exemple. En effet, si au fil du temps la technologie progresse, la fonctionnalité qu'elle supporte reste la même. Bien que le principe en lui même ne soit pas nouveau, cette approche a pour but d'automatiser ce processus. Pour exemple, on peut regarder les moyens de communication entre deux personnes. Quel que soit le moyen utilisé, la finalité reste la même : communiquer. Ainsi, dans le cadre de la téléphonie, quelle que soit l'évolution technologique ou le standard utilisé l'action de téléphoner pour un usager reste la même, composer un numéro de téléphone et attendre l'établissement de la communication. Bien entendu, une modélisation aussi simpliste que cet exemple ne saurait être une aide à la conception d'un équipement radio et requiert de multiples raffinements afin de permettre son exploitation mais l'idée commence à prendre forme.

Cette idée va être développée par l'*Object Management Group* (OMG) dès 2001 avec pour objectif d'apporter un cadre à la création de projets. Ce cadre, baptisé architecture dirigée par les modèles ou MDA (*Model Driven Architecture*), propose toute une série de standards qui permettent de définir un système. Avant de présenter plus un détail cette approche, nous allons tout d'abord présenter succinctement l'approche orientée objet sur laquelle s'appuie en partie l'approche MDA.

3.1.1 Le langage orienté objet

Une approche de conception orientée objet semble tout à fait adaptée au domaine de la radio logicielle en raison des contraintes de reconfigurabilités exigées. On privilégie même une approche dite "composant" [90]. La notion d'encapsulation, qui y est associée en général, offre un cadre adapté au changement ou remplacement d'un module de traitement comme la reconfiguration l'exige en radio logicielle. Les modélisations orientées objet présentent par conséquent un intérêt certain pour nous.

Nous n'allons pas développer ici un cours sur le langage orienté objet mais seulement en définir les bases afin de permettre la compréhension des choix que nous avons fait pour la modélisation de notre architecture. La notion d'objet permet d'extraire les données essentielles d'un système pour en synthétiser le fonctionnement. Cette synthèse, appelée objet, est représenté sur la figure 3.1.

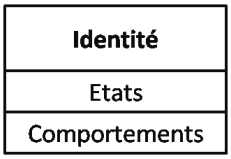


FIG. 3.1: *Représentation d'un objet*

Un objet est donc défini par une identité, un ensemble d'états et un ensemble de comportements.

- Identité : chaque objet à une existence propre, on les différencie par leur noms,
- États : valeurs instantanées des attributs d'un objet,
- Comportements : opérations possibles sur un objet.

Lorsque l'on utilise une approche orientée objet pour définir un système, on caractérise les fonctionnalités du système en le décomposant en un ensemble d'objets puis on identifie la façon dont ces objets communiquent entre eux. Ce type de représentation permet une uniformisation des interfaces de communication et permet donc d'étendre la réutilisabilité des objets définis. En effet, l'interface d'un objet est définie par les méthodes qui le composent et par lequel il est accessible. Cette approche est utilisée dans certains langages de programmation tels que C++, java, python, etc.

3.1.2 L'approche MDA

L'approche MDA développée par l'OMG, dont les spécificités sont accessibles dans [91], a été créée afin de permettre la modélisation d'un système complexe en faisant abstraction des contraintes matérielles en début de cycle de conception. MDA est illustré par la figure 3.2 extrait de [92].

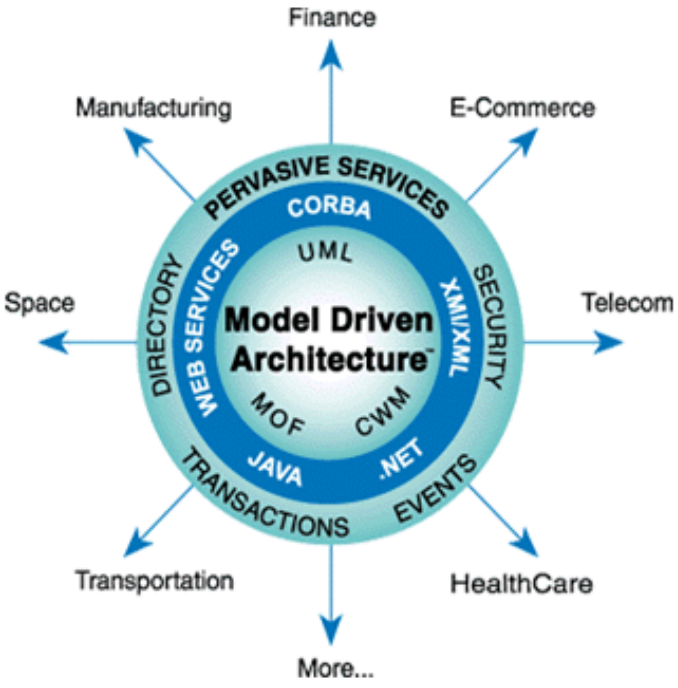


FIG. 3.2: *Représentation de l'approche MDA*

Cette illustration présente bien le principe du MDA qui est la spécification du système par un langage de haut niveau d'abstraction (positionné dans le noyau) et qui permet d'aller si on le souhaite, par passage à travers différentes couches, à la génération automatique de code dans un langage propre au domaine spécifique d'application (vers l'extérieur du cercle). On appelle langage de haut niveau d'abstraction un langage capable de synthétiser les notions complexes relatif à un domaine de compétence en les décrivant de manière abstraite.

Il existe différents niveaux d'abstraction pour la modélisation d'un système comme illustré par le tableau 3.1.

TAB. 3.1: *Différents niveaux de modélisation*

Couche	Contenu	Exemple
Meta-Métamodèle M3	Langage de description de métamodèle	Méta-Entité — 1 Source * / 1 Destination * — Méta-Relations
Métamodèle M2	Langage de description de modèle	Classe — 1 Possède * → Attributs
Modèle M1	Langage de description de données	Voiture (Type, Couleur)
Données M0	Données	Berline rouge

Il est évident que plus l'on se place à un haut niveau d'abstraction (le niveau M3 étant le plus élevé) plus l'on est indépendant de toute contrainte matérielle et plus l'on descend en abstraction plus on se lie à une cible d'exécution. MDA s'appuie sur la modélisation orientée objet et l'intègre dans une vision de modélisation orientée modèle.

On dit que l'approche MDA est orientée modèle car elle utilise quatre modèles (pouvant interagir entre eux par transformation) permettant la description d'un système complexe et de son interraction avec son environnement et dont la finalité est la génération automatique de code programme sur une cible d'exécution spécifique. En voici une brève description :

- Le modèle indépendant du calcul (CIM),
- Le modèle indépendant de la plateforme (PIM),
- Le modèle dépendant de la plateforme (PSM),
- Le code programme.

Le modèle indépendant du calcul

On appelle aussi ce modèle le modèle de métier, il définit l'environnement dans lequel le système évolue ainsi que les besoins d'un tel système. L'indépendance technologique de ce modèle lui permet de garder son intérêt au cours du temps car il n'est modifié que si les connaissances ou les besoins du métier change (ce qui n'arrive pas très souvent). Un CIM peut être exprimé dans un langage de modélisation unifié (ou UML [93] pour *Unified Modeling Language*) ou dans un langage naturel et permet donc de capitaliser les connaissances de l'entreprise.

Le modèle indépendant de la plateforme

Un modèle PIM explicite les données ainsi que le traitement des données, il représente une vue partielle du CIM. Il illustre le fonctionnement des entités et des services mais reste néanmoins indépendant des technologies utilisées pour déployer le système. Un PIM peut être exprimé à l'aide d'un langage de type UML couplé à un langage de contraintes de type OCL (*Object Constraint Language*).

Le modèle dépendant de la plateforme

Il sert essentiellement de base pour la génération de code exécutable et est fortement lié à la plateforme d'exécution du système. Le PSM définit la façon dont le système utilise la plateforme cible pour s'exécuter. Un PSM est généralement obtenu depuis un PIM par transformation de modèle suivant un ensemble de règles. Il existe des règles de refactorisation (le terme consacré étant *refactoring*) permettant de remonter à une modélisation PIM depuis une modélisation PSM.

Le code programme

Un code programme est généré à partir d'un PSM en utilisant un ensemble de règles de transformations. Le code ainsi généré respecte les contraintes spécifiques au langage cible et exploite au mieux les ressources matérielles de la plateforme cible.

Cycle de développement en Y

Ces différents modèles sont utilisés dans un cycle de conception dit cycle en Y présenté par la figure 3.3.

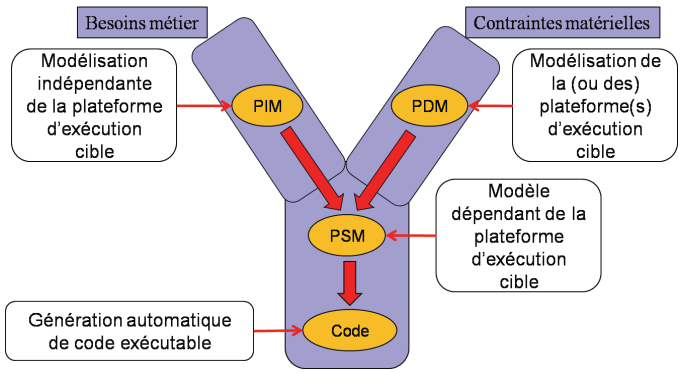


FIG. 3.3: *Cycle de conception en Y*

Les bases du MDA

L'infrastructure du MDA est définie par plusieurs standards important de l'OMG :

- le Meta Object Facility (MOF),
- le XML Metadata Interchange (XMI),
- le Common Warehouse Metamodel (CWM),
- le Unified Modeling Language (UML).

MOF

MOF [94] est un standard de métamodélisation et d'échange de construction utilisé par MDA. Les autres modèles standard de l'OMG sont définis par des constructions MOF ce qui permet de les rendre interopérables. Ce langage définit les éléments essentiels tel que la structure ou la syntaxe des métamodèles utilisés pour la construction de modèles orientés objet.

XMI

XMI [95] décrit une instance du MOF sous forme textuelle. Ce langage est le format d'échange standard entre les différents outils MDA et permet l'échange de construction entre outils de modélisation de façon uniformisée. Cet échange n'est hélas pas valide à 100%, chaque outil (voire même version d'un même outil) y allant de sa vision des choses dans la représentation d'une modélisation.

CWM

CWM [96] est un standard de gestion de données. Il définit un métamodèle qui représente les métadonnées aussi bien de niveau M3 que M1. Il est utilisé pour modéliser les échanges de métadonnées entre systèmes hétérogènes (représentation de données, analyse, gestion).

UML

UML [93] permet de modéliser un système indépendamment de toute ressource matérielle. C'est donc tout naturellement qu'il est utilisé pour la modélisation PIM mais aussi PSM par l'utilisation d'extensions à UML, appelée profile, comme nous allons le voir ci après.

3.2 Le métamodèle HDCRAM

Nous allons présenter succinctement le langage de modélisation de haut niveau UML dont nous nous sommes servis afin de modéliser notre architecture HDCRAM puis nous présenterons le résultat de cette modélisation.

UML est un langage graphique de modélisation qui permet une abstraction des systèmes complexes en un ensemble d'objets ayant diverses interactions. Cette abstraction permet d'une part, de se focaliser sur les caractéristiques principales d'une entité et d'autre part, d'être compréhensif pour des personnes non spécialistes du domaine et ce, en limitant au maximum les risques d'ambiguïté. De plus, sa représentation graphique des solutions permet une évaluation simplifiée des différentes solutions envisagées.

La prise de conscience du besoin de modéliser un système en se basant d'abord sur les savoirs de métier plutôt que sur une quelconque cible technologique ne date pas d'hier. En effet, il est reconnu depuis longtemps que la progression en terme de nouveauté est plus rapide du coté matériel que du coté logiciel. Mais qu'en serait-il si à chaque nouvelle génération matérielle les concepteurs devaient repartir de l'utilisation du silicium ? Il en résulterait automatiquement un frein dans le développement de ces technologies. En effet, depuis bien longtemps les concepteurs réutilisent des éléments logiques afin de les assembler pour créer un système complexe. Ne serait-il pas possible dans le domaine logiciel d'effectuer ce même assemblage d'éléments ?

Du début jusqu'au milieu des années 90, de nombreux langage ont été créés afin de permettre une modélisation d'éléments pérenne dans le temps et ne souffrant pas des sauts technologiques. Le problème était alors le manque de consensus autour d'une

norme de méthode d'analyse objet qui a fortement freiné l'essor de ces modélisations. Trois méthodes sur la cinquantaine de créés se sont au fur et à mesure ce sont imposées : OMT, Booch et OOSE. UML est le fruit de l'unification et de la normalisation de ces trois approches par l'OMG. La première version d'UML fut validée en 1997, actuellement, l'OMG a standardisé UML 2.1.2 depuis novembre 2007.

3.2.1 Modélisation UML

Une modélisation UML est orientée objet et s'appuie sur une représentation possible d'un système au travers d'un certain nombre de diagrammes. Avant d'effectuer une description sommaire de ces diagrammes et de leur utilisation, nous allons poser les notions essentielles à la compréhension de cette modélisation.

La modélisation UML s'appuie sur l'utilisation de classes permettant de regrouper un ensemble d'éléments ayant des caractéristiques communes. Afin d'appréhender cette notion, il est possible de s'imaginer un arbre généalogique avec les notions d'héritage et de relation. La figure 3.4 illustre le principe d'héritage entre classes.

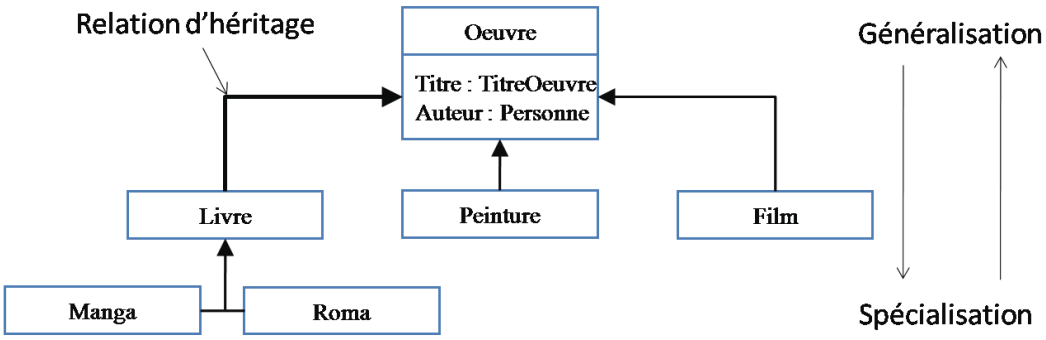


FIG. 3.4: *Représentation des relations d'héritage*

La classe Oeuvre est appelée classe parent, les classes reliées à cette classe par relation d'héritage sont appelées classe enfant et hérite des attributs ainsi que des méthodes définit dans la classe parent. Ainsi, bien que cela ne soit pas stipulé, la classe Manga possède aussi un titre et un auteur mais ayant été défini dans la classe parent, il n'est nul besoin d'y faire à nouveau mention. Il est à noter que la notion d'instanciation de classe fait référence à la création d'un objet auquel on a attribué des valeurs, pour l'exemple précédent, une instanciation de la classe roman pourrait être : Harry Potter, Joanne Kathleen Rowling.

L'une des justifications à la prolifération du nombre de diagrammes utilisables dans une modélisation UML est qu'il y a différents moyen de modéliser un système suivant ce que l'on veut mettre en valeur. En effet, la modélisation permet de mettre en évidence les points important d'un système afin de ne pas noyer la personne qui en fait la lecture

sous un flux d'information trop important et non pertinent. On dénombre ainsi 13 diagrammes de modélisation qui sont proposés par UML, ceux ci peuvent être classés en trois groupes :

- Les diagrammes structurels (présentés par le tableau 3.2),

TAB. 3.2: *Diagrammes structurels*

Type de diagramme	Objectifs
Diagramme de classes	Représenter les classes intervenant dans le système
Diagramme d'objets	Représenter les instances de classe
Diagramme de composants	Identifier les interfaces de communication entre composants (accès bibliothèque, base de données …)
Diagramme de déploiement	Représenter les éléments matérielles, la plateforme d'exécution cible ainsi que les relations entre eux
Diagramme des paquetages	Un paquetage représentant un ensemble d'éléments, un tel diagramme illustre les dépendances entre ces ensembles
Diagramme de structure composite	Représenter la structure interne d'une classe et les collaborations qui permettent à cette classe de fonctionner

  Cet ensemble de diagramme permet d'illustrer les divers éléments composants un système complexe, leur interaction via leur interface ainsi que de modéliser la façon dont les éléments vont utiliser les ressources de la plateforme matérielle cible.

- Les diagrammes comportementaux (présentés par le tableau 3.3),

TAB. 3.3: *Diagrammes comportementaux*

Type de diagramme	Objectifs
Diagramme des cas d'utilisation	Représenter les interactions possible entre le système et les acteurs
Diagramme état-transition	Représenter sous forme de machines d'état le comportement du système ou de ses éléments
Diagramme d'activité	Représenter l'enchaînement des actions du système ou de ses éléments suite à une action

  Ces trois types de diagramme permettent de montrer de quelle façon le système ou un élément réagit lorsqu'une action est engagée.

- Les diagrammes d'interactions (présentés par le tableau 3.4).
  Les diagrammes d'interactions illustrent les principes de fonctionnement d'un élément avec un autre élément ou un de ses propres composants.

Bien entendu, il n'est pas nécessaire d'avoir recours à l'ensemble de ces diagrammes afin de modéliser un système. Certains sont plus utiles dans des secteurs bien précis (un diagramme de déploiement est par exemple particulièrement adapté pour le secteur de

TAB. 3.4: *Diagrammes d'interactions*

Type de diagramme	Objectifs
Diagramme de séquence	Représenter séquentiellement l'exécution des traitements et les interactions entre les éléments du système et/ou de ses acteurs
Diagramme de communication	Représenter les échanges de message entre objets (version simplifiée d'un diagramme de séquence)
Diagramme global d'interaction	Variante du diagramme d'activité
Diagramme de temps	Représenter l'évolution d'une donnée au cours du temps

l'électronique moins pour le secteur financier). Il existe de très nombreux outils permettant la création de diagrammes UML, hélas, très peu d'entre eux sont conforme à 100% aux spécifications UML (notamment la version 2) ce qui ne permet pas par exemple la création de l'ensemble des graphes de façon optimisée à partir d'un seul outil. Un autre problème survient aussi lors de l'échange de modélisation entre outil qui peuvent s'avérer être incompatible. Les outils les plus connus sont Rational ou encore Together, on peut aussi nommer un plugin gratuit, pour l'environnement de développement Eclipse [97], Eclipse UML2.

Notion de profile

Le profile UML est une notion importante de la modélisation UML qui a été introduite depuis la version 1.3 d'UML. Un profile UML fournit les mécanismes permettant de spécialiser une description UML à un contexte particulier de travail comme par exemple le codage, l'analyse ou encore la conception. À ce titre, un profile introduit des notions plus adaptées à un type de traitement : règles de modélisation spécifique, règles de production de documentation ou encore mode de présentation. Prenons l'exemple d'un profile permettant la génération de code VHDL à partir d'une description UML. Ce profile UML "VHDL" va définir les extensions nécessaire à l'établissement d'une conception UML pour VHDL ; vérifier que le modèle UML respecte les contraintes VHDL ; produire un code VHDL conforme aux règles définies par le standard VHDL par application de règles de transformation adéquates. Un profile peut aussi être utilisé dans le cadre d'une modélisation de bases de données, de représentation/spécification d'architecture ou tout autre contexte de travail. Intégré à un outil de conception, un profile permet donc de bénéficier d'un guide d'assistance, de vérifications ou encore d'automatisation (de génération de code, de génération de documentation . . .) adaptés à un cadre d'étude.

Cette notion de profile est un moyen permettant de structurer les besoins particuliers de chaque domaines modélisés par UML. Il existe ainsi de nombreuses extensions regroupées en profile UML représentant des standards tels que UML pour CORBA, UML pour les EJB (*Enterprise Java Beans)*) permettant d'apporter les spécificités relatives à ces domaines précis d'application comme illustré par la figure 3.5.

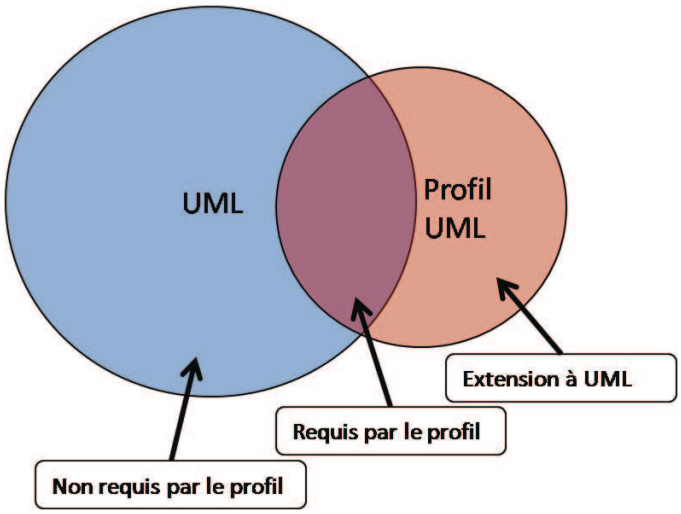


FIG. 3.5: *relation entre UML et un profile UML*

Un profile UML peut hériter d'un autre profile, avoir des dépendances avec un autre profile ou être regroupé avec d'autres profiles. SOFTEAM, qui est en charge de diriger ce travail au sein de l'OMG, regroupe dans [98] les informations exprimées par un profile UML comme suit :

- Extensions ajoutées : Présentation des extensions à UML
- Règles de validation : Permet la vérification des critères de cohérence sur un modèle pour un profile donné par un ensemble de règles définies.
- Règles de présentation : Permet la génération automatique de diagrammes ou filtre sur les diagrammes utilisable.
- Règles de transformation : Permet la définition des produits de développements, des règles de génération de code et des design patterns permettent d'assister ou d'automatiser le développement spécifiquement pour chaque type d'activité.

Utilisation d'UML pour l'ingénierie système

UML est devenu un standard de modélisation dans le monde logiciel et est maintenant vu comme suffisamment robuste pour aider à la modélisation de systèmes d'ingénierie complexe. Par l'OMG, qui garantit une forte participation industrielle, le langage UML se voit doté de spécifications pour la définition, l'analyse, la conception et la vérification de systèmes complexes. Ces spécifications ont pour but de participer à l'amélioration de l'échange de modèles entre outils de conception et de combler le fossé sémantique entre les différentes disciplines impliquées dans cette modélisation. En 2001, des discussions entre l'*International Council on Systems Engineering* (INCOSE) et l'OMG ont mené à la décision de créer un groupe responsable de l'UML pour les systèmes d'ingénierie (UML SE pour *Unified Modeling Language for Systems Engineering*) et de le

raccorder à l'OMG. Ce groupe nommé *Systems Engineering Domains Special Interest Group* ou SE DSIG fit son premier meeting le 13 septembre 2001 à Toronto. Les premiers travaux accomplis par ce groupe furent une réflexion sur les moyens de mise en œuvre et les besoins d'une modélisation UML pour l'ingénierie des systèmes ainsi que la participation à la version 2 d'UML. Ces travaux, présentés en 2003 dans [99], apportent les conclusions menant à l'établissement d'un standard suivi par l'OMG afin de certifier le suivi via une évolution en accord avec les partenaires industriels. La poursuite de ces travaux amena à la création d'un langage de modélisation système nommé SysML (*Systems Modeling Language*) [100] qui a vu sa version 1 validée en septembre 2007.

3.2.2 Métamodèle HDCRAM

La recherche présentée précédemment sur les langages de modélisation à haut niveau s'inscrit dans notre démarche de réflexion, comme présenté dans la troisième ligne de la figure 4 résumé par la figure 3.6, et dont le but est la modélisation d'HDCRAM par un langage de haut niveau d'abstraction.

Objectives	Means	Results
Symbolic representation of the reality	Language (UML)	HDCRAM Metamodel

FIG. 3.6: *Modélisation de HDCRAM*

Le choix de la modélisation UML pour HDCRAM se justifie facilement par la nature pluridisciplinaire de la radio cognitive ainsi que l'étendue et la nature hétérogène du parc d'équipement visé. Il est en effet essentiel de fournir un modèle qui :

- ne soit pas affecté par les sauts technologique,
- permette de synthétiser les besoins de chaque corps de métier intervenant dans la conception d'équipement RC,
- permette de passer outre les contraintes des langages dédiés qui ne sont interprétable que par des experts d'un domaine en particulier,
- permette de capitaliser les savoirs acquis.

Nous proposons dans cette section de présenter HDCRAM par différents diagrammes UML afin de mettre en avant ses fonctionnalités et ses interactions. Pour ce faire nous présenterons tout d'abord l'architecture dans sa globalité avec un diagramme de classe, puis nous présenterons comment circulent les données dans HDCRAM. Ensuite nous présenterons rapidement les interactions de HDCRAM avec son environnement puis nous détaillerons les unités ReM et CRM.

Modélisation HDCRAM

Afin de permettre une lecture aisée de notre architecture et d'illustrer les dépendances des gestionnaires hiérarchique distribués nous avons eu recours à une modélisation UML de notre architecture. Cette modélisation a uniquement pour but de décrire fonctionnellement notre architecture de gestion cognitive d'un équipement radio à un niveau d'abstraction élevé. Le but est de définir l'ensemble des constituants nécessaires au bon fonctionnement de l'équipement.ainsi que la liste des opérations qu'ils doivent pouvoir effectuer. Dans ce sens, notre modélisation correspond à une description PIM. Cette modélisation s'appuie sur l'utilisation d'un diagramme de classe (qui est généralement reconnu comme l'élément central d'UML) et est présentée par la figure 3.7. La vue de l'architecture présentée ici est entièrement centrée sur le gestionnaire et son contrôle sur le système et met en avant les dépendances ainsi que les cardinalités entre gestionnaire.

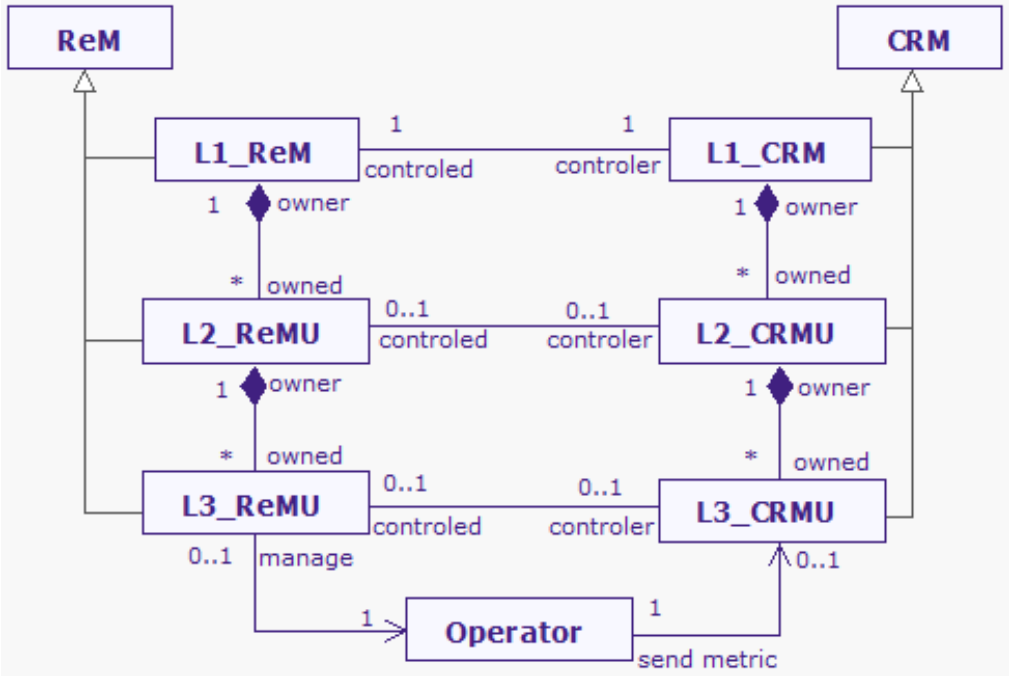


FIG. 3.7: *Métamodèle de l'architecture HDCRAM*

Les classes présentes sur le coté gauche de l'architecture sont les niveaux de gestion de reconfiguration et sur la droite les niveaux de gestion cognitive. La classe Operator, présente en bas du métamodèle, est une classe parent qui regroupe l'ensemble des opérateurs pouvant être implantés dans l'équipement radio. La partie haute de la figure met en évidence les classes parent ReM et CRM qui illustrent la généricité des fonctionnalités des unités de gestion déployées dans l'architecture. Il est cependant important de garder à l'esprit que ces fonctionnalités seront spécialisées suivant le traitement qui sera attribué aux classes enfant.

À la lecture de la figure 3.7 il apparaît sans ambiguïté possible les relations de dépendance entre les gestionnaires composant HDCRAM. Ainsi il est clairement établi que :

- si une unité ReM existe, alors une unité CRM peut lui être associée (et inversement),
- il ne peut y avoir présence d'une entité L3 sans entité L2 pour la contrôler,
- une entité CRM contrôle son entité ReM associée,
- pour une entité de niveau supérieur (CRM ou ReM) il existe de 0 à une infinité d'unités de niveau inférieur associées.

Flot de données dans HDCRAM

La figure 3.8 illustre de manière globale les flux d'informations entre unités du gestionnaire.

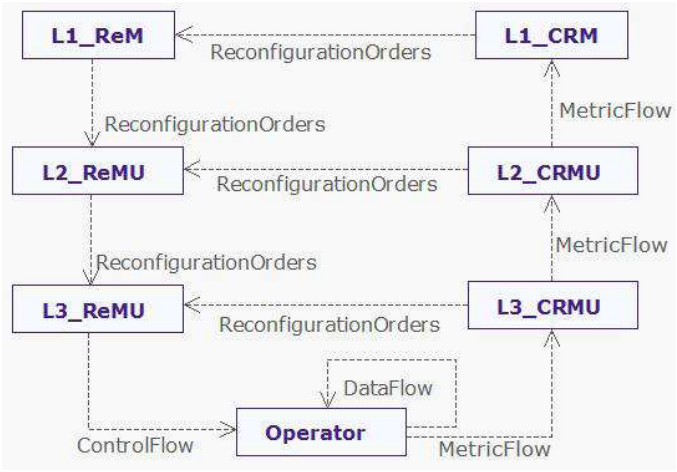


FIG. 3.8: *Flux d'information dans l'architecture HDCRAM*

On illustre ainsi de façon claire ce qui a été présenté par la figure 2.11 au chapitre deux à savoir

- une approche "top/down" pour caractériser l'envoi d'ordre de reconfiguration dans la partie ReM,
- une approche "bottom/up" pour illustrer la remontée de métrique dans la partie CRM,
- une approche transversale (d'une unité CRM vers son unité associée ReM) pour l'envoi d'ordre de reconfiguration décidé par un gestionnaire cognitif.

Relations de HDCRAM avec l'environnement

Bien qu'autonome dans sa gestion de plateforme d'exécution, l'architecture HDCRAM réagit à son environnement afin de décider d'une reconfiguration de l'équipement. HDCRAM ne définit pas de façon intrinsèque les politiques de sécurité ou d'utilisation de spectre, nous considérons en effet que ces entités ne bénéficieraient pas d'une intégration au sein de l'architecture. Néanmoins, il faut prévoir un mode d'accès en lecture par HDCRAM à ces politiques afin que la solution de reconfiguration de l'équipement, préconisée par HDCRAM, soit en concordance avec les politiques en vigueur dans son environnement. De plus, il faut aussi prévoir l'accès en écriture à ces mêmes politiques afin de venir effectuer des modifications si besoin est. On prendra comme exemple un changement de pays ou les standards d'émission/réception changent. Dans ce cas, HDCRAM fera une demande de confirmation des politiques en vigueur et si de nouveaux paramètres lui sont envoyés il mettra le module à jour. Il est clair que cette liberté laissée à HDCRAM nécessite des précautions pour l'accès à ces règles en termes de sécurité pour ne pas perturber d'autres utilisateurs. Le diagramme 3.9 présente les relations d'HDCRAM avec son environnement.

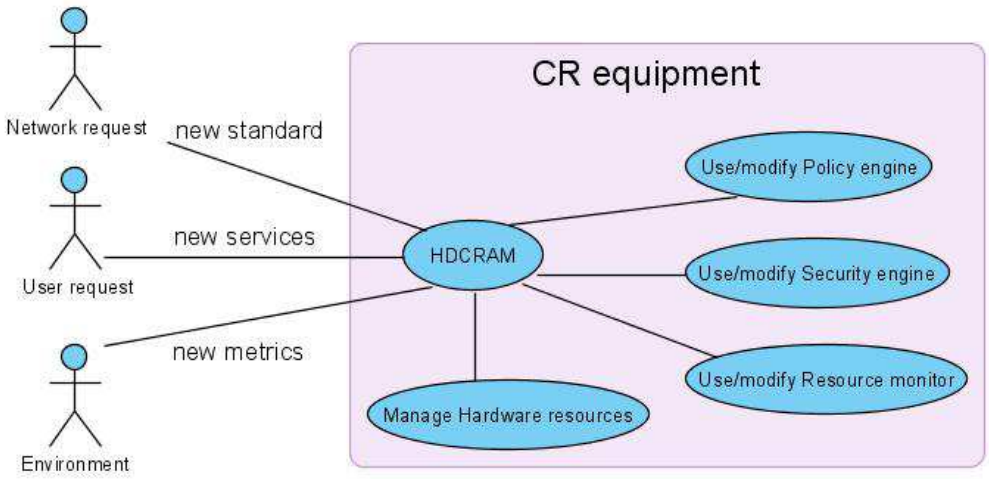


FIG. 3.9: *Diagramme des cas d'utilisation de l'architecture HDCRAM*

Réaction d'une unité CRM

Afin de détailler le fonctionnement interne d'une unité de gestion cognitive, nous avons utilisé ce que l'on appel un diagramme de données-processus (*process-data diagram*) qui permet de lier le parcours d'une donnée à un ensemble de processus. Ce diagramme illustré par la figure 3.10 permet de faire une corrélation entre les actions nécessaires à l'exécution d'un système cognitif (à gauche) et les différentes opérations (à droite) au sein d'un Li_CRM(U) (i pouvant prendre une valeur de 1 à 3). Les différentes

opérations ainsi que les dépendances les reliant sont ici mise en évidence. Il est à noter que les opérations présentées ci-dessus sont utilisées par chaque niveau d'abstraction. En effet, le cycle cognitif est répété pour chaque Li_CRM(U). Néanmoins, le traitement des métriques et les actions qui en découlent offrent des répercussions différentes suivant le niveau qui les déploie.

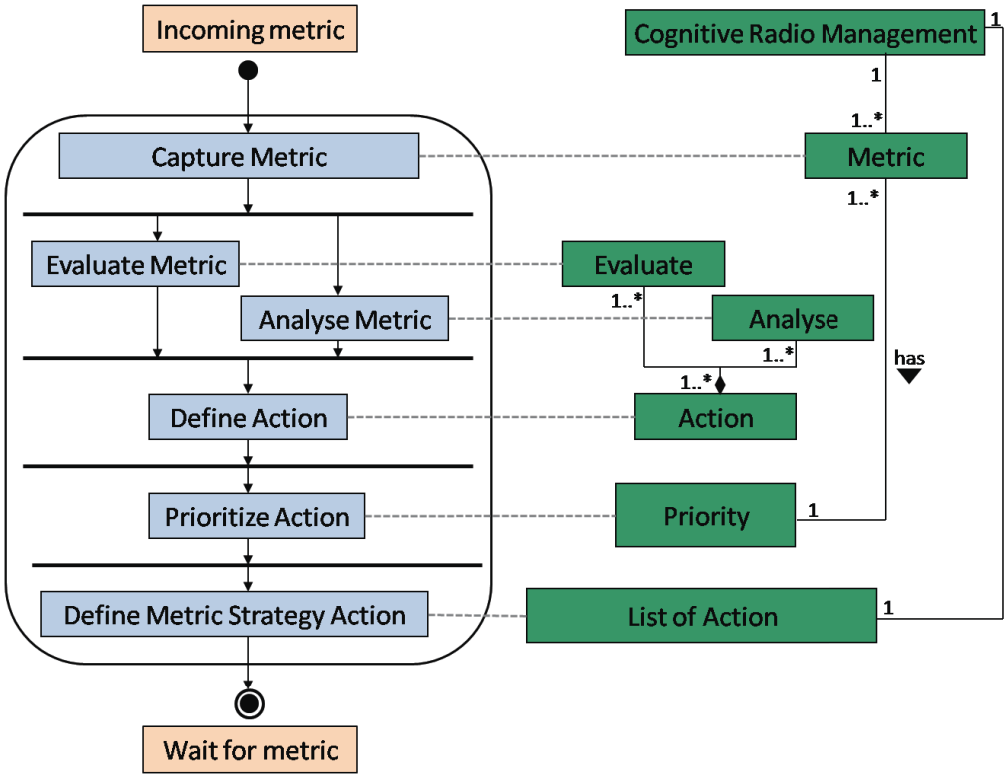


FIG. 3.10: *Représentation UML des opérations d'une unité CRM*

On remarquera que la partie gauche est identique à la figure 2.12. La partie droite permet de représenter les besoins en termes d'opération et de ressource au sein d'une entité CRM. Ainsi il apparaît qu'une unité CRM possède une unique liste d'actions lui permettant de définir les statégies d'actions pour la mise en place d'une reconfiguration. Cette liste d'action peut être fournie par exemple par les politiques d'utilisation du spectre en cours dans la localisation de l'équipement radio. L'unicité de cette liste d'actions permet de contraindre le gestionnaire dans ses degrés de liberté, bien entendu elle peut bénéfier de l'auto apprentissage ou encore être modifiée par une mise à jour. De même quelle que soit le nombre de métriques collectées par une unité CRM, chacune possède une priorité permettant de définir son importance dans une prise de décision de reconfiguration.

Réaction d'une unité ReM

De même, nous illustrons ici avec la figure 3.11 les relations entre l'interprétation d'un ordre de reconfiguration d'une unité CRM associée ou ReM de niveau supérieur et le déclenchement des processus au sein d'une unité ReM.

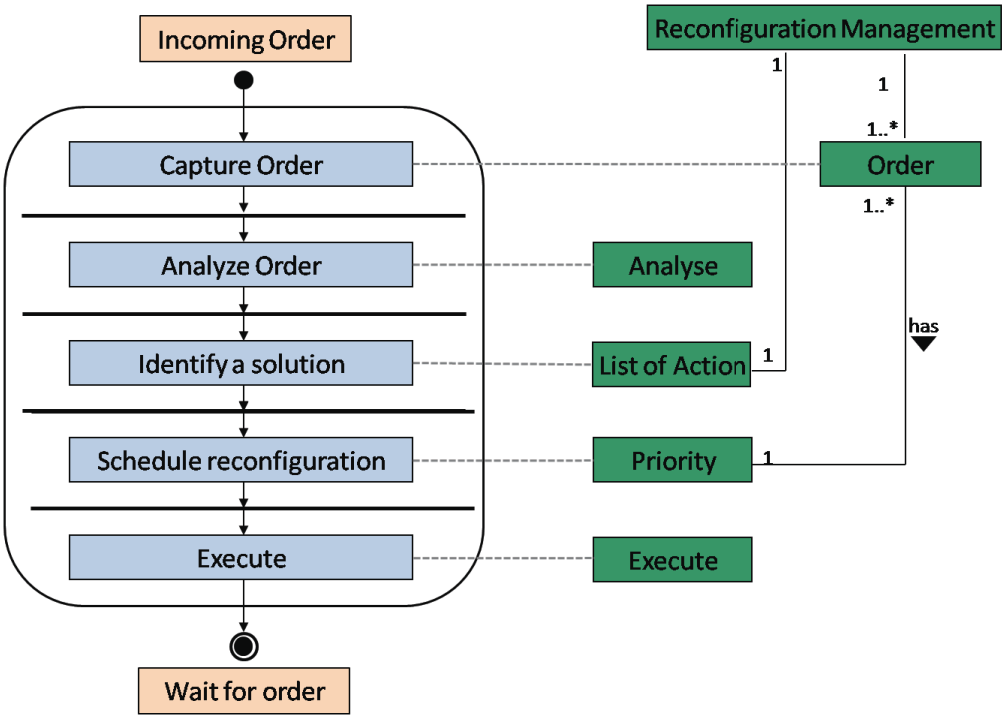


FIG. 3.11: *Représentation UML des opérations d'une unité ReM*

- Capture order : définit l'action d'acquisition d'un ordre de reconfiguration, qu'il vienne du CRMU associé ou du ReMU associé de niveau supérieur,
- Analyze order : vérifie la validité de la donnée,
- Identify a solution : Interprète l'ordre de reconfiguration en fonction des paramètres de configuration courant,
- Schedule reconfiguration : Analyse la priorité et la répercute dans sa table d'ordonnancement des tâches,
- Execute : Exécute la reconfiguration.

Apparaît dès lors deux problématiques : la première est comment mettre en place une politique de gestion de priorité d'ordre de reconfiguration pour éviter un conflit dans un CMU entre un ordre venant de son CRU et un ordre venant du niveau supérieur. La seconde est la mise en place de moyens permettant d'éviter les boucles infinies d'optimisation. Ces deux problèmes dépassent le cadre de cette thèse mais nécessitent une part non négligeable de travail d'exploration pour la mise en place d'un système cognitif.

Il est à noter que les opérations présentées dans les deux diagrammes ci-dessus représentent une partie seulement des opérations contenues dans une unité de gestion. En effet, les opérations de type rafraîchissement des valeurs, passage de valeur, mise à jour des bibliothèques, etc. ne sont pas détaillées ici.

Résultat de la modélisation

Il est évident que cette thèse n'a pas vocation à délivrer un outil permettant la création automatique d'un équipement RC clé en main. L'outil que nous proposons ici permet une exploration fonctionnelle visant à identifier tous les éléments de gestion qu'il est nécessaire d'intégrer dans un équipement RC. Ceci afin que cet équipement puisse supporter l'ensemble des scénarios envisagés. La liste des opérations que doivent être capable d'exécuter ces éléments et les éventuelles bases de données nécessaires sont aussi identifiée grâce à l'outil. À l'échelle de cette thèse, seule une étude purement fonctionnelle était visée. Néanmoins, il est envisageable à l'avenir, de coupler cette modélisation HDCRAM avec un profile tel que le profile MARTE [101] ayant pour vocation l'aide à la Modélisation et l'Analyse des Systèmes Temps Réel et Embarqués. Bien entendu, l'intérêt de l'intégration de ce profile se fera à un niveau d'abstraction moindre puisque ce profile permet de prendre en considération les aspects matériels de la plateforme d'exécution cible. Ce type d'approche est notamment retenue dans le projet MOPCOM [102] qui est un projet du "pôle de compétitivité Images & Réseaux" de la région Bretagne de trois ans (2007-2010). Ce projet a pour vocation la production d'une méthodologie complète ainsi que d'un environnement de développement adapté à la conception de systèmes embarqués sur des systèmes sur puce (SoC). Cet environnement de développement doit pouvoir accompagner un concepteur de système reconfigurable depuis la modélisation haut niveau de son application jusqu'à la génération automatique de code programme VHDL ou SystemC.

La force de ce métamodèle HDCRAM est qu'il n'est en rien lié uniquement aux systèmes RC mais peut être utilisé pour la conception de tout équipement présentant des facultés d'auto-reconfiguration. À cet effet, aucune obligation n'a été apportée sur l'utilisation de tel ou tel algorithme cognitif (il peut en effet être extrêmement simpliste tout comme se révéler extrêmement complexe) ni même sur le langage servant à en définir le comportement. Suivant le domaine d'utilisation fixé, un concepteur pourra réutiliser ce métamodèle et définir (ou réutiliser) des profiles, quelque soit le domaine d'application, afin de l'accompagner jusqu'à la génération de code programme.

3.2.3 Vers un métamodèle exécutable

Nous avons donc modélisé notre architecture à l'aide du langage UML, ce qui permet une spécification formalisée, claire et précise de l'architecture et une compréhension qui ne requiert pas d'être un expert dans le domaine (avec il est vrai le besoin d'avoir des notions UML). Néanmoins, une des limitations du langage UML est qu'il ne permet pas de simuler de façon fonctionnelle le système modélisé. Il n'est utilisé que dans l'optique de modéliser des concepts complexes au niveau structurel (ce qui est déjà loin d'être trivial) et d'apporter une vue plus générale des concepts utilisés. Il faut par la suite utiliser des transformations de modèles afin de pouvoir simuler ces concepts et valider l'approche retenue. S'il s'avère que le code programme généré ne fournit pas les résultats escomptés il va falloir identifier à quel niveau les spécifications n'ont pas été assez finement déclarées. Une solution serait alors de pouvoir effectuer une simulation bien avant la génération de code. Dans l'optique d'effectuer des simulations de concept dans le cadre d'une modélisation PIM, il est nécessaire de requérir l'utilisation d'un langage de métaprogrammation exécutable. Ce type de langage permet d'ajouter une description comportementale à une description structurelle des objets afin d'effectuer des simulations fonctionnelles du système modélisé. Cette étape de modélisation, dont le résultat est appelé métamodèle exécutable, s'incrit dans le troisième objectif de cette thèse représenté à la quatrième ligne de la figure 4 résumé par la figure 3.12.

Objectives	Means	Results
Verification/Refinement of the functionalities	**Design specific language (Kermeta)**	**HDCRAM Simulator**

FIG. 3.12: *Définition d'HDCRAM à l'aide d'un métalangage de programmation*

L'avantage de l'utilisation d'un langage de métaprogrammation exécutable est qu'il peut être écrit indépendamment de toute cible matérielle. De plus il peut être utilisé afin de générer le code programme. De cette façon on évite tout problème relatif à la cohabitation de plusieurs langages hétérogènes pour la définition d'un système comme décrit en Annexe 1. Notre approche s'inscrit donc dans une approche dite IDM (Ingénierie Dirigée par les Modèles) qui ne se limite plus à l'unique utilisation de métamodèles MOF/UML mais s'ouvre à d'autres langages de métamodélisation.

Le langage que nous avons retenu pour l'élaboration du métamodèle exécutable de l'architecture HDCRAM est Kermeta [103], [104]. Kermeta est développé à l'INRIA de Rennes par l'équipe Triskell, il dispose d'un environnement de développement de métamodèles basé sur EMOF (Essential Meta Object Facility) [105] et est parfaitement intégré à l'environnement Eclipse. Kermeta permet non seulement de décrire la structure des métamodèles, mais aussi leur comportement. Il permet ainsi de définir et d'outiller

de nouveaux langages spécifiques à un domaine (DSL : Domain Specific Language) en améliorant la manière de spécifier, simuler et tester la sémantique opérationnelle des métamodèles. Cette démarche respecte parfaitement l'approche MDA puisque Eclipse utilise le langage Ecore pour la construction de métamodèle et que ce dernier est une version comparable à MOF. De plus, il est possible d'effectuer une transformation des constructions vers un fichier textuel XMI ce qui permet une interopérabilité avec d'autres logiciels de modélisation. Plus de précisions sur Kermeta sont fournies en Annexe 1.

Il pourrait sembler antinomique de requérir à la définition d'un langage dédié à un certain domaine alors que l'on veut définir un système à un très haut niveau d'abstraction. Néanmoins, ayant pour objectif la définition d'un métamodèle exécutable pour un équipement RC nous sommes déjà dans une optique de restriction du champ d'exploration. Ainsi, nous visons à restreindre les libertés offertes par UML afin de nous focaliser sur une utilisation pour un contexte RC. Il est à noter que la restriction est moins importantes que si nous avions eu recours à un langage de type C ou VHDL. Ce langage pourra être par la suite enrichi par des spécificités de chaque domaine de recherche (algorithmiques, logicielles, matérielles ...) et ainsi constituer un nouveau profile dédié aux équipements RC.

Il existe un langage créé pour la RC par J. Mitola nommé RKRL (*Radio Knowledge Representation Language*). Ce langage permet de rendre interprétable par toute machine RKRL, ou compatible RKLR, les données de son environnement et son fonctionnement propre dans le temps et l'espace. L'utilisation de ce langage permet à un équipement de comparer les données de l'environnement avec les modèles qu'il possède et d'en déduire les paramètres de fonctionnement optimal. RKRL va bien plus loin qu'un langage de modélisation et ne peut être utilisé dans le cadre de cette thèse. En effet, l'utilisation de ce langage permet de changer le comportement de l'équipement par modification des paramètres opérationnels, mais il permet aussi d'interpréter les besoins des utilisateurs exprimés en langage naturel afin de satisfaire ses demandes. Pour illustrer cela, on peut reprendre l'exemple suivant extrait de [2].

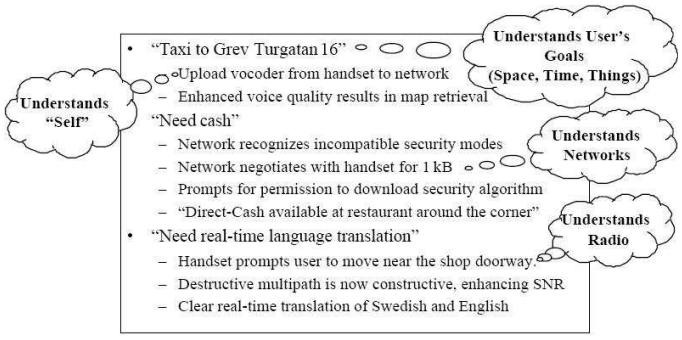


FIG. 3.13: *Service avancé d'aide à l'utilisateur*

Sur cet exemple on voit clairement que le langage RKRL permet de comprendre les besoins des utilisateurs et d'y répondre suivant les données relevées dans son environ-

nement (localisation, heure, date ...). L'un des problèmes évident de ce type de langage naturel est l'ambiguïté et la complexité que représente son interprétation par une machine. Bien que ce genre de technologie commence à faire son apparition notamment dans un moteur de recherche internet développer par Powerset (récemment racheté par Microsoft) ou encore le fournisseur de services mobiles Chacha (qui permet de poser vocalement n'importe quelle question et d'obtenir une réponse unique par message texte) elle est encore loin d'être mature. De plus le langage RKRL n'a pas été conçu dans l'optique de contrôler les ressources matérielles en fonction de leurs spécificités propres et son utilisation requiert une connaissance certaine de ce langage.

3.3 Le simulateur HDCRAM

Nous avons conçu et développé un simulateur de déploiement et de comportement d'HDCRAM pour un équipement RC. L'objectif est de valider les principes de cette architecture générique (tant du point de vue structurel que comportemental) pour tous les scénario RC imaginables. La fusion des architectures déployées pour tous ces scénarios constituerait l'ensemble des possibilitées d'architecture que doit pouvoir intégrer un équipement RC.

3.3.1 Création du métamodèle exécutable HDCRAM à l'aide de Kermeta

Nous avons intégré le métamodèle présenté par la figure 3.7 dans l'environnement de développement Eclipse avec le plug-in Kermeta. Le résultat de cette modélisation de diagramme de classe est sauvegardé en *diagram.ecore*. De ce fichier, le concepteur peut générer automatiquement par transformation un fichier d'extension Kermeta : *diagram.kmt*. Depuis ce fichier le concepteur va définir la structure comportementale de son système et définir son DSL. La figure 3.14 illustre en haut un exemple de classe avec une définition classique (i.e. le cœur des opérations n'y est pas décrit). En bas de cette figure apparaît le comportement de l'opération *hello* lorsqu'elle est appelée au travers de la classe Test$_k ermeta$.

Pour exemple, la figure 3.15 suivante présente une partie du code Kermeta spécifiant la structure comportementale de notre métamodèle.

Le code présenté offre le choix à l'utilisateur de déployer un scénario depuis le début (instanciation de toutes les classes nécessaires au bon déroulement du scénario) ou de charger une sauvegarde précédemment effectuée. Selon le choix exprimé, une opération sera appelée : *load* pour charger ou *run* pour déployer un scénario.

```
class Test_abstract
{
    attribute name : kermeta::standard::String

    operation hello() : Void is
    abstract
}

class Test_kermeta
{
    attribute name : kermeta::standard::String

    operation hello() : Void is do
        stdio.writeln("hello world !")
    end
}
```

FIG. 3.14: *Spécification de la sémantique comportementale*

Une fois l'aspect comportemental défini, une nouvelle transformation est effectuée vers un fichier ecore qui sera exécutable. L'exécution se fera dans une console utilisateur permettant de s'assurer du bon fonctionnement du système ainsi modélisé. Si des problèmes persistent, après avoir été identifiés à l'aide de la simulation, le métamodèle peut être raffiné par modification du code Kermeta jusqu'à satisfaction du concepteur.

La figure 3.16 illustre les moyens de raffiner un métamodèle exécutable exprimé à l'aide du langage Kermeta.

Il apparaît sur cette figure les différentes étapes de conception d'un métamodèle exécutable, de la spécification graphique (se faisant sous forme de diagramme de classe UML) jusqu'à la validation finale.

3.3.2 Résultat : un métamodèle exécutable

La figure 3.17 présente de façon partielle le simulateur que nous avons mis au point pour effectuer les simulations nous permettant de préciser la structure de l'architecture HDCRAM ainsi que la sémantique opérationnelle.

```
class HDCRAM_simulator
{
    attribute hdcram2l1_crm : L1_CRM[1..1]#l1_crm2hdcram
    attribute hdcram2l1_rem : L1_ReM[1..1]#l1_rem2hdcram

    operation main() : Void is
        /* Entry to HDCRAM_simulator */
        do
            var sim : StandardReference init "init"
            stdio.writeln(" ")
            stdio.writeln("HDCRAM_simulator : main")
            sim := stdio.read("Launch an existing simulation or create new ? (l/c)")
            if ((sim != "l") and (sim != "c")) then
                sim := stdio.read("Bad entry : Launch an existing simulation or create new ? (l/c)")
            end
            if (sim == "l") then
                load
            else
                run
            end
        end
```

FIG. 3.15: *Code Kermeta spécifiant un aspect comportemental de la classe HDCRAM_simulator*

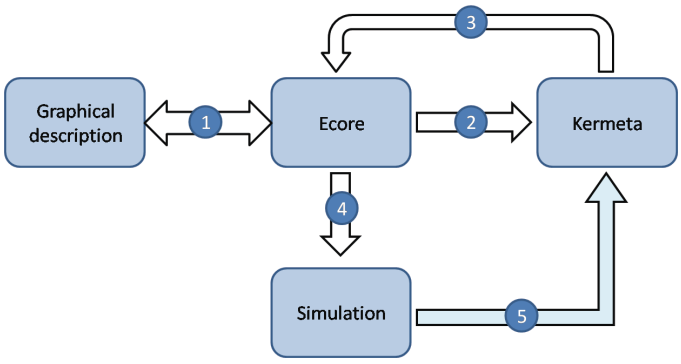


FIG. 3.16: *Étape de raffinement du métamodèle HDCRAM*

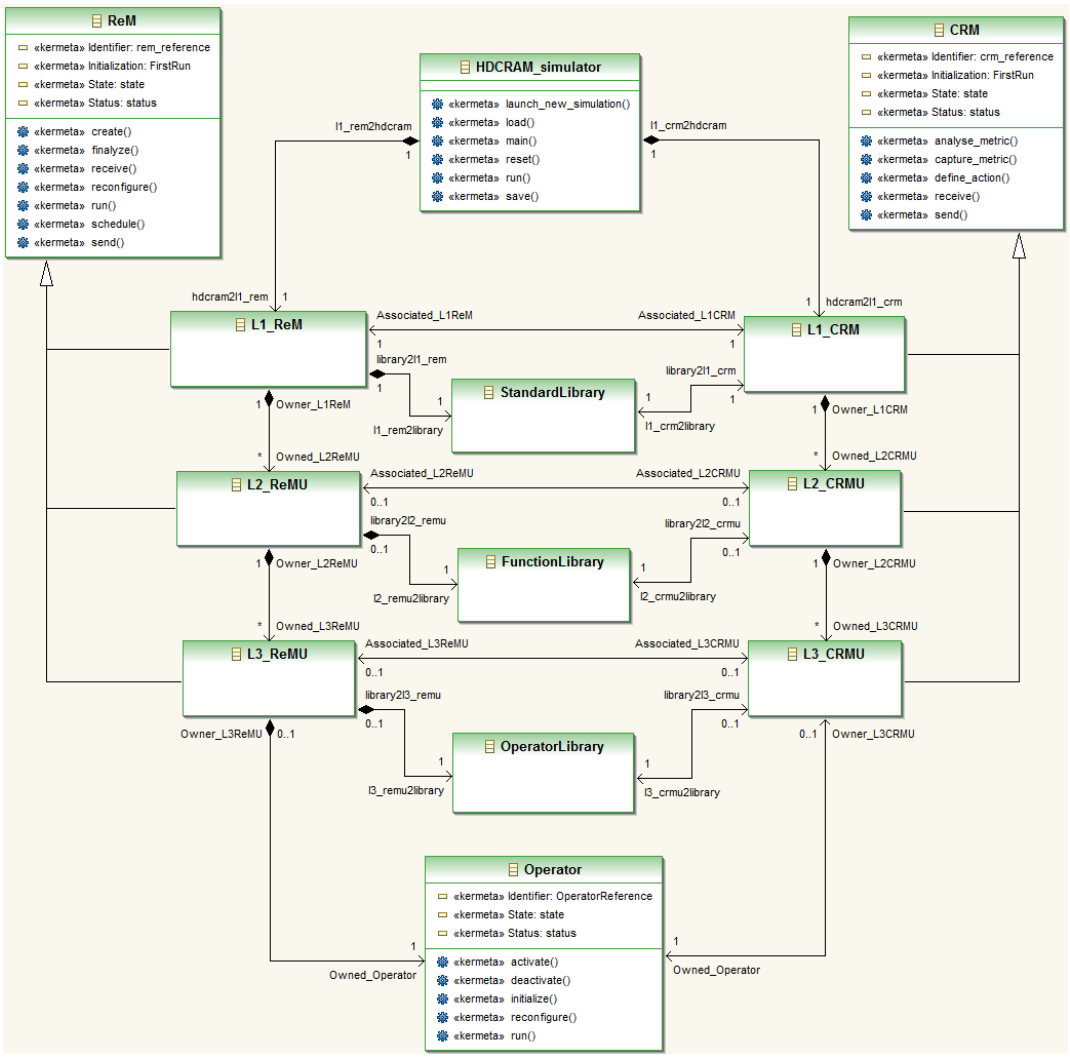


FIG. 3.17: *Simulateur de l'architecture HDCRAM*

On y retrouve l'architecture présentée dans 3.7 avec de nouvelles propriétées. Afin de permettre l'exécution de simulations à partir du métamodèle, nous avons mis en place une classe nommée HDCRAM_simulator. Cette classe est définie de manière à permettre la simulation fonctionnelle de l'architecture, c'est elle qui créer la classe L1_ReM et lance la simulation. Elle permet aussi la sauvegarde de l'état des classes instanciées pour une simulation donnée et permet ainsi l'exécution d'une simulation préalablement sauvegardée. Lors d'une exécution, l'utilisateur choisit un scénario à tester puis valide la création des entités nécessaires par la visualisation de l'architecture déployée. Une fois l'architecture déployée, l'utilisateur est invité à rentrer un certain nombre de métriques qui vont entraîner des modifications sur l'état du système. Ainsi il sera possible de visualiser ces modifications et de valider les prises de décision. La partie centrale de l'architecture présentée sur la figure 3.17 met en avant les librairies accessibles aux différentes entités CRM(U) et ReM(U).

Ces librairies renseignent le système sur les moyens à mettre en œuvre pour constituer les différentes chaînes de traitement et contiennent des informations comme :

- La configuration actuelle,
- les fichiers binaires exécutables (nécessaires à la reconfiguration des opérateurs sur FPGA ou DSP),
- le taux d'utilisation des ressources matérielles,
- la charge des processeurs,
- etc.

Lors de la mise en place d'un nouveau standard par exemple, l'entité L1_ReM consulte la bibliothèque StandardLibrary afin de connaître le nombre et le nom des L2_ReMU à créer (en fonction de ceux déjà en place) ainsi que les paramètres de fonctionnement à leur attribuer.

Les différentes librairies sont mises à jour (par le réseau ou par auto-apprentissage) par création (ou modification) de nouveaux fichiers d'opérateurs ou de nouveaux algorithmes de traitement. À chaque couple CRM/ReM une unique bibliothèque est associée. La partie auto-apprentissage est représentée par les unités CRM qui peuvent modifier les bibliothèques afin de prendre en compte leur expérience de reconfiguration de paramètres opérationnels.

3.3.3 Classes ReM

La figure 3.18 illustre la classe parent ReM. Chaque unité ReM déployée dans l'architecture hérite de ses attributs ainsi que de ses opérations.

Une classe héritant de la classe parent ReM est identifiée par les attributs et opérations présentés par le tableau 3.5 :

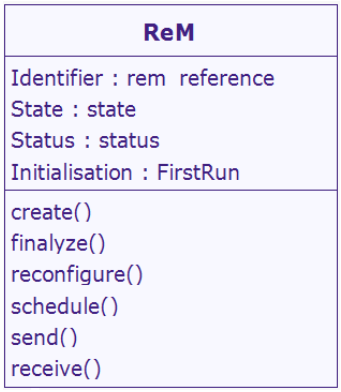


FIG. 3.18: *Super classe ReM*

En fonction du niveau de hiérarchie où sera instanciée la classe enfant, de nouveaux attributs ou opérations spécifiques peuvent être créés comme l'identification des unités ReM qui lui sont associées (si elle existe, l'unité ReM dont elle dépend ainsi que l'(les) unité(s) ReM qui dépend(ent) d'elle). De plus les méthodes dérivées des super méthodes peuvent être affinées suivant les besoins. Par exemple l'opération *create()* utilisée par une classe ReM enfant peut générer différentes sorties suivant son niveau de hiérarchie :

- L1 :create => un ou plusieurs L2_ReMU,
- L2 :create => un ou plusieurs L3_ReMU,
- L3 :create => un opérateur.

Les tableaux 3.6, 3.7 et 3.8 suivants présentent les attributs ,utilisés par les ReM(U) suivant leurs niveaux de hiérarchie, nécessaires et suffisants pour leur permettre d'assurer le bon fonctionnement d'HDCRAM. Par exemple, on voit dans le tableau 3.8 qu'un L3_ReMU ayant la charge d'implanter l'opérateur de traitement doit connaître la cible d'exécution. À ce titre, il dispose de l'attribut Target. De manière générale, une unité L3_ReM à une unité CRM associée, ceci est indiqué par l'attribut Associated_L3CRMU. Il est à noter que dans un soucis de généralisation comme expliqué précédemment, chaque unité ReM possède une unité CRM associée comme illustré dans ces tableaux.

TAB. 3.5: *Description de la classe parent ReM*

	Attributs	Définition
ReM	Identifier	Identifiant du gestionnaire
	Initialization	Phase d'initialisation ou phase normale
	State	État de l'unité (traitement, bogue, reconfiguration en cours, etc.)
	Status	Objectif à atteindre fixé par l'unité ReM de hiérarchie supérieure ou CRM associée
	Opérations	**Définition**
	Create	Création d'une unité ReM de hiérarchie inférieure ou d'un opérateur ou invocation d'une unité CRM associée
	Finalize	Destruction une unité ReM de hiérarchie inférieure ou un opérateur ou une unité CRM associée
	Reconfigure	Mise en forme d'un ordre de reconfiguration pour unité ReM de hiérarchie inférieure ou pour l'opérateur à charge
	Schedule	Ordonnancement d'une reconfiguration dans la table des tâches
	Send	Envoi d'informations de reconfiguration
	Receive	Réception d'ordres de reconfiguration de l'unité ReM de hiérarchie supérieure ou de l'unité CRM associée

TAB. 3.6: *Attributs spécifiques à L1_ReM*

Unité	Attributs	Descriptif
L1_ReM	Associated_L1CRM	Unité CRM associée
	L1_rem2hdcram	Connexion entre le L1_ReM et le HDCRAM_simulator
	Standard_reference	Regroupe l'ensemble des L2_ReMU contrôlées

TAB. 3.7: *Attributs spécifiques aux L2_ReMU*

Unité	Attributs	Descriptif
L2_ReMU	Associated_L2CRMU	Unité CRM associée
	Function_reference	Regroupe l'ensemble des L3_ReMU contrôlées
	Owner_L1rem	Connexion avec le ReM de hiérarchie supérieure

TAB. 3.8: *Attributs spécifiques aux L3_ReMU*

Unité	Attributs	Descriptif
L3_ReMU	Associated_L3CRMU	Unité CRM associée
	Bug detected	Problème sur l'opérateur contrôlé
	Owner_l2remu	Connexion avec le ReM de hiérarchie supérieure
	Target	Cible matérielle d'exécution

3.3.4 Classes CRM

La figure 3.19 illustre la classe parent CRM. Chaque unité CRM déployée dans l'architecture hérite de ses attributs ainsi que de ses opérations.

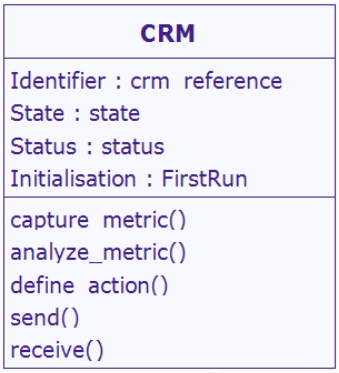


FIG. 3.19: *Super classe CRM*

Une unité CRM est responsable du processus de décision de reconfiguration de l'équipement. En accord avec la définition des sciences informatiques, un agent logiciel a le pouvoir de décider quelle action est nécessaire et quand. Ceci correspond parfaitement au rôle assigné aux unités CRM dans HDCRAM. En effet, chaque unité CRM agit comme un agent logiciel au sens ou chaque unité est consciente de son objectif à atteindre et des capacités des éléments qu'elle contrôle. La force de HDCRAM est l'organisation de ces agents logiciels en architecture hybride. Ce type d'architecture à pour but d'organiser un ensemble d'agents intelligents dans une hiérarchie de manière à combiner un comportement dirigé par les buts avec un comportement réactif au changement de l'environnement. Il existe de nombreuses architectures hybride définies dans le monde logiciel. L'une des plus connues est InteRRaP (*Integration of Reactive Behavior and Rational Planning* [106], [107]. Cette architecture, dont le nom peut être traduit en français par "intégration du comportement réactif et planification rationnelle", est une architecture en couche verticale ou les données d'entrées (telles que les métriques de perception) passent successivement d'une couche à l'autre. On remarque ici le parallèle qui peut être fait avec HDCRAM et sa hiérarchie dans la distribution des unités CRM. La différence se fait sur les sorties possible de solution dans chacune des trois couches de hiérarchie. Ceci permet ainsi de boucler le cycle cognitif plus rapidement de manière à offrir plus de réactivité au système. Nous pouvons dire que les unités intelligentes, distribuées dans le système multi-agent HDCRAM, peuvent être vues comme des agents hybrides réagissant à des stimuli extérieurs et prenant des décisions suivant leurs propres déductions.

Une classe héritant de la classe parent CRM est identifiée par les attributs et opérations présentés par le tableau 3.9 :

TAB. 3.9: *Description de la classe parent CRM*

	Attributs	Définition
CRM	Identifier	Identifiant du gestionnaire
	Initialization	Phase d'initialisation ou phase normale
	State	État de l'unité (traitement, bogue, reconfiguration en cours, etc.)
	Status	Objectif à atteindre fixé par l'unité ReM associée
	Opérations	**Définition**
	Capture_metric	Acquisition d'une ou plusieurs métriques
	Analyze_metric	Analyse de la ou des métriques reçues
	Define_action	Décision d'une reconfiguration ou non et mise en œuvre
	Send	Envoi d'informations de reconfiguration à l'unité ReM associée ou envoi de métriques à l'unité CRM de hiérarchie supérieure
	Receive	Réception d'une ou plusieurs métrique(s) de(ou des) l'unité(s) CRM de hiérarchie inférieure ou de l'opérateur

De nouveaux attributs, comme l'identification des unités CRM qui lui sont associées [i.e. si elle existe, l'unité CRM dont elle dépend ainsi que l'(les) unité(s) CRM qui dépend(ent) d'elle], peuvent être attachés à une classe enfant CRM suivant ses spécificités. Une unité CRM est responsable de l'interprétation des variations de l'environnement, de l'optimisation des ressources matérielles ainsi que de la prise de décision de reconfiguration. Après une prise de décision, une unité CRM transmet l'ordre de reconfiguration à son unité ReM associée et attend la validation de ses ordres par le relevé de métrique de son (ou ses) unité(s) controlée(s). Les nouveaux paramètres opérationnels sont transmis à l'unité de hiérarchie supérieure (si elle existe), une fois la reconfiguration validée. Dans le cas d'un problème lors d'une reconfiguration, une unité CRM peut effectuer le chargement d'une sauvegarde préalable afin de revenir dans un état connu et lance une procédure lui permettant d'analyser la cause du problème et d'en tirer les enseignements. Une unité CRM est invoquée ou détruite suivant les ordres de son unité ReM associée. Le terme "invocation" est ici choisi car il me permet de bien mettre en valeur la nature intelligente de l'entité CRM. À contrario on créer une unité CRM qui n'est à proprement parlé qu'un objet qui exécute les données qu'il reçoit.

Les tableaux 3.10, 3.11 et 3.12 suivants présentent les attributs utilisés par les CRM(U) suivant leurs niveaux de hiérarchie.

3.3.5 Super Classe Operator

La classe parent Operator est illustrée par la figure 3.20.

TAB. 3.10: *Attributs spécifiques à L1_CRM*

Unité	Attributs	Descriptif
L1_CRM	Associated_L1ReM	Unité ReM associée
	Function_name	Regroupe l'ensemble des L2_CRMU contrôlées
	L1_crm2hdcram	Connexion entre le L1_CRM et le HDCRAM_simulator
	L1-crm2library	Connexion entre le L1_CRM et sa bibliothèque
	L2_crmu_metric	Regroupe l'ensemble des objectifs associés aux L2_ReMU

TAB. 3.11: *Attributs spécifiques aux L2_CRMU*

Unité	Attributs	Descriptif
L2_CRMU	Associated_L2ReMU	Unité ReM associée
	L2_crmu2library	Connexion entre le L2_CRMU et sa bibliothèque
	L3_crmu_id	Regroupe par identifiant l'ensemble des L3_CRMU contrôlées
	L3_crmu_status	Regroupe les métriques des L3_CRMU contrôlées
	Owner_L1crm	Connexion avec le CRM de hiérarchie supérieure

TAB. 3.12: *Attributs spécifiques aux L3_CRMU*

Unité	Attributs	Descriptif
L3_CRMU	Associated_L3ReMU	Unité ReM associée
	L3_crmu2library	Connexion entre le L3_CRMU et sa bibliothèque
	Owned_operator	Renseigne sur l'opérateur contrôlé
	Owner_l2crmu	Connexion avec le CRM de hiérarchie supérieure

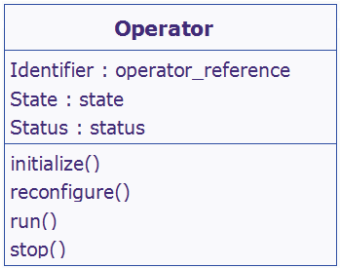


FIG. 3.20: *Super classe Operator*

Comme précisé précédemment, nous considérons ici qu'un opérateur traite les données qu'il reçoit en entrée et produit des données en sortie. Sa reconfiguration est contrôlée par son unité ReM associée et il produit des métriques pour son unité CRM associée.

Étant dans une démarche de modélisation à haut niveau, nous ne représentons un opérateur que par son interface qu'il présente au reste de l'architecture en termes d'opérations. Il n'est donc nullement fait mention de la cible d'exécution (FPGA, DSP, GPP) ou de paramètres spécifique d'exécution (puissance de calcul, temps de traitement, consommation, etc.). Les fonctionnalités retenues pour la modélisation d'un opérateur sont illustrées dans le tableau 3.13.

TAB. 3.13: *Description de la classe parent Operator*

	Attributs	Définition
Operator	Identifier	Identifiant de l'opérateur
	State	État de l'opérateur (traitement, bogue, reconfiguration en cours etc.)
	Status	Métrique à usage de l'unité CRM en charge
	Opérations	**Définition**
	Initialize	Remise à zéro de l'opérateur (reset matériel ou logiciel)
	Reconfigure	Reconfiguration de l'opérateur
	Run	Traitement de l'opérateur
	Stop	Arrêt de l'opérateur

3.3.6 Déploiement du métamodèle exécutable

Dans un premier temps, il faut déployer le système, cette phase sera effectuée en début de vie du système et ne sera plus utilisée par la suite. Le déploiement se compose de trois phases, la création du niveau L1 puis celle du niveau L2 et enfin la création du niveau L3.

Déploiement du niveau 1

Un standard ainsi qu'une performance initiale sont choisi. Le choix à ici été pris de considérer le matériel comme étant dans les meilleures conditions, les meilleures performances seront donc employées. Il est bien évidemment possible de stipuler les conditions initiales diféremment. Lors de cette première phase de déploiement, le L1_ReM et le L1_CRM sont créés. Le lien l1_rem_controler connectant ces deux entités va permettre un échange afin que le L1_CRM identifie son L1_ReM. Le L1_ReM charge à partir de sa bibliothèque de données le nombre ainsi que l'appellation des L2_ReMU à instancier pour le type de standard qui a été défini.

FIG. 3.21: *Déploiement du niveau 1*

Déploiement du niveau 2

Puis le L1_ReM instancie les L2_ReMU. A chaque L2_ReMU créé, un L2_CRMU est invoqué. Ils communiquent par leur lien l2_crmu_controler afin que la partie cognitive se connecte à la partie management. Les L2_CRMU envoient leurs paramètres courants au L1_CRM. Les L2_ReMU instanciés accèdent à la bibliothèque de fonction afin de connaître le nombre et le nom des différents L3_ReMU qu'ils devront créer.

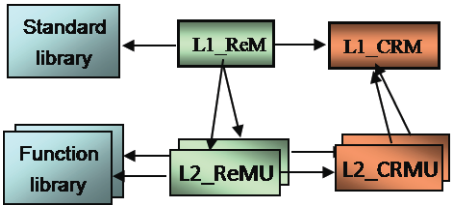


FIG. 3.22: *Déploiement du niveau 2*

Déploiement du niveau 3

Chaque L2_ReMU instancie donc le nombre adéquat de L3_ReMU afin de réaliser la fonction désirée. Ces L3_ReMU communiquent via le lien l3_crmu_controler avec leur L3_CRMU respectif. Puis les L3_CRMU envoient leurs paramètres à leur L2_CRMU respectif. Ensuite, chaque L3_ReMU accède à la bibliothèque d'opérateurs et, suivant le nom et la performance de l'opérateur à instancier un binary file ou un bitstream est chargé

afin d'implanter l'opérateur sur cible (DSP, FPGA ...). Ce déploiement est illustré par la figure 3.23.

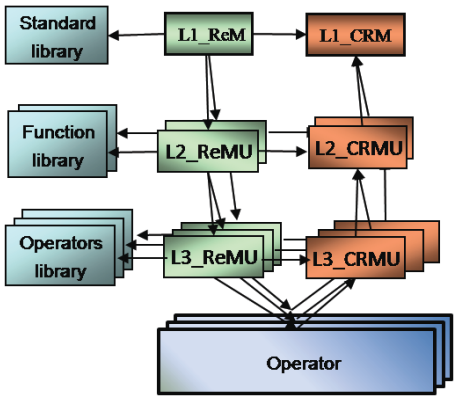


FIG. 3.23: *Déploiement du niveau 3*

Chaque opérateur ainsi créé envoie à son L3_CRMU un signal d'activité permettant de valider la bonne implantation.

Une fois le système déployé, on rentre dans la phase normale de traitement. Le système devra répondre de la manière la plus adaptée aux différents stimuli auxquels il sera soumis. À une fréquence donnée (suivant l'importance de la métrique et son évolution au cours du temps) les opérateurs indiqueront à leur L3_CRMU qu'ils sont prêts à envoyer une métrique. Dans l'état actuel de notre simulateur, cet envoie de métrique est exécuté par une saisie au clavier de l'utilisateur pour chaque opérateur utilisé dans le scénario. Chaque L3_CRMU effectue alors une lecture de cette métrique et l'interprète. Cette interprétation ce fait sur une "gamme" d'évolution précise de la métrique. Un exemple serait le niveau de batterie du système. Si la métrique est entre 75 et 100 % alors cela se traduira par une valeur "haute". Entre 50 et 75 % par une valeur "moyenne", entre 25 et 50 % par une valeur "basse" et enfin entre quelques pourcents et 25 % par une valeur "très basse". Au niveau L3, le degré de liberté est très restreint puisque les possibilités de prise de décision sont limitées au seul opérateur. Dans la majorité des cas un L3_CRMU effectuera donc uniquement une interprétation de la métrique et la remontera à son L2_CRMU. Il faut néanmoins prévoir la possibilité d'accorder à un L3_CRMU, via son L3_ReMU associé, la reconfiguration de l'opérateur à charge afin d'effectuer du "bug-fixing". Lorsque un L2_CRMU évalue et analyse les métriques de ses L3_CRMU il effectue une interprétation globale sur cet ensemble. Au niveau L2 les possibilités de reconfiguration deviennent plus importantes. Bien qu'il soit possible d'utiliser un L2_CRMU pour relever la(ou les) métrique(s) d'un seul L3_CRMU, il est évident que dans la majorité des cas celui-ci sera associé à plusieurs L3_CRMU. Tout dépendra du niveau de granularité choisi pour réaliser la fonction et de la réutilisation des opérateurs. Un L2_CRMU, via son L2ReMU associé, pourra ordonner la reconfiguration d'un ou plusieurs L3_ReMU afin de modifier le comportement des opérateurs dont les métriques sont relevées par les L3_CRMU à charge. Les L2_CRMU fournissent

chacun une métrique au L1_CRM qui les interprète et défini une action de reconfiguration soit pour un changement de standard soit pour “équilibrer” les performances d’une ou plusieurs fonctions.

3.4 Conclusion

Nous avons présenté dans ce chapitre HDCRAM qui engloble HDReM pour la gestion de la reconfiguration en y ajoutant les capacités d’une gestion fine des prises de décision par ses entités CRM(U) hiérarchiques et distribuées. De plus HDCRAM intègre efficacement les nouveaux opérateurs de type capteur afin de permettre une prise de conscience des variations dans l’environnement de l’équipement RC.

Le métamodèle exécutable, tel qu’il est défini actuellement est une architecture de support à la prise de décision et n’intègre pas de puissants algorithmes de décision. Il permet de souligner de manière claire les mécanismes inhérents à une prise de décision pour la reconfiguration tels que :

- Comment sont captées les métriques ?
- Comment sont elles traitées ?
- Comment sont exécutés les ordres de reconfiguration au sein du système ?

En effet nous rappelons que dans le cadre de cette thèse, les objectifs sont de définir de manière **fonctionnelle** les besoins d’une architecture de gestion pour un équipement RC. Nous nous sommes contentés ici de modéliser les unités de gestion par des boîtes noires faisant l’acquisition de données et produisant, en réaction, des données. Ainsi les mécanismes de prise de décision et de mise en œuvre de reconfiguration sont ici exprimés de façon rudimentaire. Ceci ouvre donc de nombreuses pistes de recherches tant dans le contenu des fonctions de prise de décision que des moyens d’implantations (recherche et installation des fichiers d’implantation) et du contrôle de composants implantés sur cible.

Nous avons aussi présenté les raisons du choix d’une modélisation UML pour notre HDCRAM. Nous avons présenté cette norme déjà incontournable dans le monde logiciel et qui est envisagée de plus en plus dans le domaine de la conception matérielle. En effet, ce standard assure une capitalisation des connaissances “métier” des entreprises dans leur domaine d’activité par une opportunité de réutilisation du savoir faire en s’affranchissant des sauts technologiques. De plus cette modélisation permettant une spécification à haut niveau de systèmes complexes étant standardisée par une organisation indépendante, elle assure une pérennité des créations ainsi qu’une compréhension facilitée et sans ambiguïté d’un système complexe par des personnes d’expertises différentes. Ce langage s’adapte donc parfaitement à la description d’architecture pour les équipements RC qui sont par nature hétérogènes. De cette modélisation nous avons déduit qu’il était possible d’offrir

plus en permettant à un concepteur d'effectuer une simulation de son modèle HDCRAM par l'intermédiaire du métalangage de programmation Kermeta. Nous avons donc créé un métamodèle exécutable que nous avons nommé HDCRAM_simulator. Dans la troisième section de ce chapitre, nous avons introduit les fonctionnalités de ce métamodèle exécutable ainsi que son principe de déploiement. Nous avons aussi introduit les étapes nous ayant permis de raffiner HDCRAM_simulator. Il est important de noter que la version du simulateur proposée ici ne se veut pas experte dans la prise de conscience de son environnement ou dans ses prises de décision. En effet, il appartient à un niveau inférieur d'abstraction d'introduire les algorithmes permettant à un équipement RC d'acquérir l'intelligence nécessaire à son fonctionnement tel qu'il a été décrit par J. MITOLA. Ce simulateur souligne le fonctionnement de l'architecture ainsi que les moyens de la mettre en œuvre.

Nous allons introduire dans le chapitre 4 des cas d'étude permettant de souligner ce fonctionnement et de présenter le simulateur à travers des résultats de simulation.

CHAPITRE

4 Scénarios

Sommaire

4.1 Introduction

NOUS avons présenté HDCRAM avec tout d'abord l'introduction de son gestionnaire de reconfiguration lui permettant d'assurer des reconfigurations multi-granularité et multi-standard. Puis nous avons présenté le gestionnaire cognitif qui offre de supporter les fonctions permettant à un équipement radio de prendre conscience à la fois de son environnement mais aussi de ses propres capacités opérationnelles de fonctionnement. Ces deux gestionnaires réunis au sein de la même architecture permettent à un équipement RC de s'adapter aux variations de l'environnement, aux variations inhérentes à son fonctionnement (baisse du niveau de batterie, bogue, etc.) ainsi qu'aux demandes de l'utilisateur et du réseau. Nous avons conçu un simulateur HDCRAM présenté précédemment dans la section 3.3 du chapitre 3 dont l'intérêt se situe, pour notre utilisation, dans l'exploration de scénario et le raffinement de l'architecture elle même en fonction des résultats d'exécution. À termes, ce simulateur pourra être proposé afin de permettre la compréhension des mécanismes de HDCRAM voire la simulation à très haut niveau d'équipements RC pouvant mener à la génération automatique de code programme. Nous proposons d'illustrer ici l'intérêt de son utilisation dans le cadre de la conception d'un équipement RC.

Dans ce chapitre nous allons présenter dans un premier temps, un scénario où le relevé d'une métrique impliquera le court-circuitage d'un opérateur ainsi que son arrêt. Ce type de solution peut être utilisé pour améliorer la vitesse de traitement dans un cadre

optimal de travail ou lorsqu'une économie d'énergie est à mettre en place. Le deuxième scénario justement présentera le cas d'une métrique relevant le niveau de batterie. Cette information entraînera dans un premier temps un ajustement des paramètres opérationnels permettant une économie de puissance. Dans un deuxième temps, l'opérateur pourra être réalloué sur une nouvelle cible matérielle permettant l'utilisation d'une IP moins consommatrice en énergie. Le troisième et dernier scénario présentera un cas d'utilisation où un équipement seul prendra conscience de son environnement afin d'utiliser un standard de façon opportuniste.

Nos études d'autres scénarios nous ont montré que ces trois scénarios balayent l'ensemble des cas de gestion RC d'un équipement. Il est à noter que nous ne présenterons pas dans ce chapitre la phase de déploiement des divers éléments participant au bon déroulement du scénario, cette phase ayant déjà été présentée dans la sous section 3.3.6 du chapitre 3. Une fois la simulation lancée, l'utilisateur est invité à entrer les données utiles à la simulation au travers une interface textuelle permettant aussi de contrôler l'exécution des différentes phases par affichage des sorties du simulateur. Un exemple de cette interface textuelle attendant une entrée utilisateur est présenté par la figure 4.1.

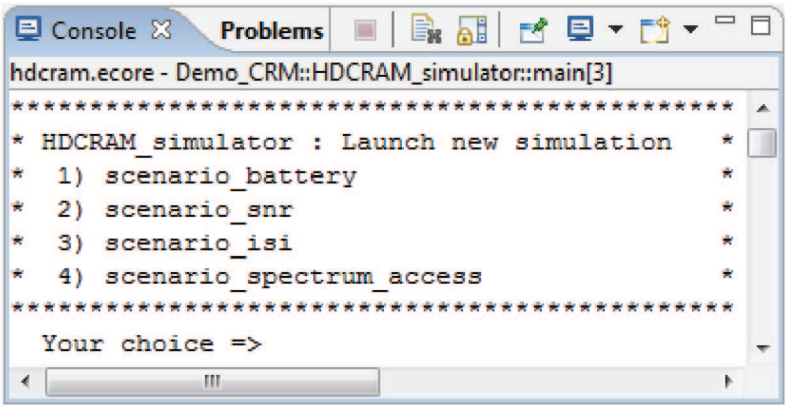


FIG. 4.1: *Exemple d'invite de commande*

Il est à noter que pour une raison de lisibilité des figures illustrant l'architecture déployée, nous ne représenterons pas les bibliothèques rattachées à chaque couple CRM/ReM bien que celles ci soient bien présentes et utilisées dans les diverses scénario que nous allons présenter. De plus, l'objectif des scénarios présentés ici étant d'insister sur les mécanismes de gestion inhérent à HDCRAM (tant au niveau gestion de reconfiguration que de gestion cognitive) il ne sera fait mention que des éléments participant à l'établissement du scénario présenté. Afin de faciliter la présentation, on appelera module L1 le couple L1_ReM/L1_CRM, module L2 un couple L2_ReMU/L2_CRMU et module L3 un couple L3_ReMU/L3_CRMU.

4.2 Présentation de l'environnement de travail

Avant d'introduire des résultats de simulation, nous allons tout d'abord présenter succintement l'environnement de travail par l'intermédiaire de la figure 4.2 ci après.

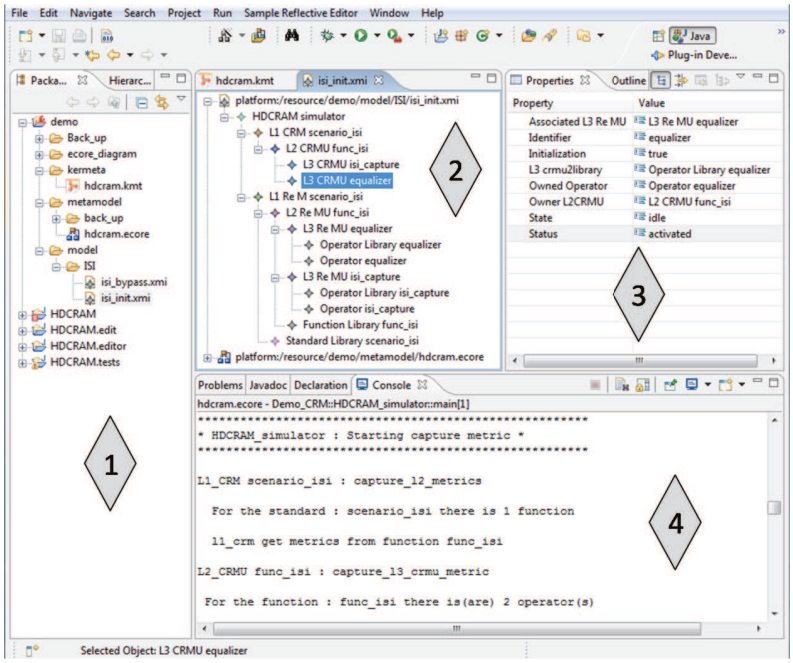


FIG. 4.2: *Présentation de l'environnement de travail*

Cette figure présente quatre zones distinctes numérotées de 1 à 4 dans les diamants. La zone 1 représente avec une vue arborescente le répertoire de travail du projet. On y trouve principalement :

- un dossier kermeta contenant le fichier décrivant la structure du métamodèle ainsi que son comportement,
- un dossier metamodel renfermant l'interprétation du métalangage de programmation Kermeta en fichier ecore (qui on le rappel est une version comparable à MOF utilisée dans Eclipse)
- un dossier model contenant les sauvegardes des exécutions de simulation.

La zone 2 présente de manière arborescente un fichier XMI. Couplée avec l'utilisation de la zone 3, cette vue graphique du fichier XMI permet d'accéder simplement aux propriétés des attributs ou dépendances d'un objet instancié. La zone 4 est l'interface du simulateur, celle à partir de laquelle un utilisateur entre de nouvelles instructions et vérifie leurs effets sur l'architecture.

la figure 4.3 illustre comment sont répartis les différents objets suivant leur niveau de hiérarchie.

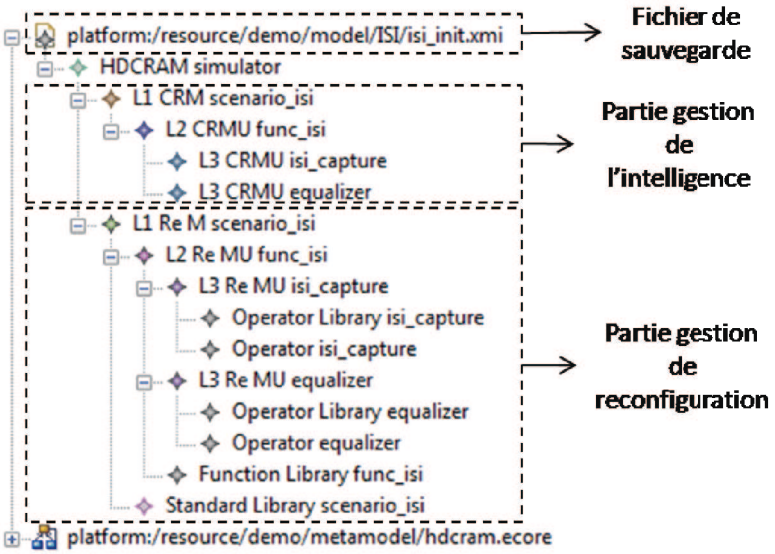


FIG. 4.3: *Vue globale de l'architecture déployée*

Il est évident que la composition de l'architecture (nombre d'unités déployées) dépend du scénario joué. Il est difficile à partir de cette vue d'assurer qu'il existe des dépendances entre unité CRM et ReM, voire que les bibliothèques (visible grâce à l'étiquette *Library*) ont des liens avec les unités CRMU. Néanmoins il aurait fallu bénéficier d'une représentation 3D afin de tout faire paraître et il n'est pas évident que cela aurait facilité la lecture des éléments de l'architecture. On note cependant une similitude dans les appelations comme par exemple :

- L3_CRMU **equalizer**,
- L3_ReMU **equalizer**,
- Operator Library **equalizer**,
- Operator **equalizer**.

4.3 Scénario1 : Reconfiguration d'un chemin de données

Le premier scénario présenté ici illustre un cas d'étude où il est possible de court-circuiter un opérateur et d'arrêter son fonctionnement lorsqu'il n'est plus nécessaire. Par exemple, lors de l'émission d'un signal radio, un phénomène présentant un grand nombre de trajets de propagation apparaît. Ce phénomène, appelé propagation par trajets multiples, peut être lié aux réflections causées par le rebond des ondes radio sur les bâtiments par exemple. Les problèmes liés aux trajets multiples sont généralement la génération d'ISI (pour *Intersymbol Interference*) qui sont caractérisés par l'apparition d'interférences ainsi qu'une perturbation du signal reçu. En effet une multitude de signaux

arrivent au récepteur qui doit reconstituer le signal d'origine. Dans la plupart des cas, les effets générés par l'ISI sont minimisés par l'utilisation d'égaliseurs linéaires (utilisant par exemple un critère de forçage à zéro ou d'erreur quadratique moyenne minimum). Nous n'allons pas nous intéresser ici aux moyens de combattre ce phénomène inhérent aux communications numériques mais illustrer la possibilité d'adaptation d'un équipement utilisant HDCRAM en fonction de l'importance de ce phénomène. Cette adaptation peut se faire via l'utilisation ou non d'égalisation sur le signal reçu en fonction de la force de l'interférence.

L'ISI pouvant être mesurée (par un estimateur de canal par exemple), l'intérêt de l'utilisation ou non de l'égaliseur peut être sujet à la valeur de cette mesure. Ainsi, on peut estimer que si la valeur de l'ISI relevée est inférieure à 10%, le canal peut être désigné comme propre et l'égalisation n'est plus nécessaire.

Notons que le court-circuitage d'un opérateur de traitement peut avoir d'autres intérêts comme l'augmentation de la vitesse globale de traitement du signal ou encore la possibilité d'effectuer un compromis entre qualité du signal reçu et économie d'énergie. L'exemple présenté ici se rapporte à la participation ou non de l'opérateur égaliseur mais peut s'appliquer à d'autres opérateurs "non essentiel" au traitement du signal, à un moment donné, en fonction du context d'opération de l'équipement.

Comme illustré par la figure 4.4, l'architecture déployée pour ce scénario est simple puisqu'elle se résume ici à deux opérateurs : un capteurs d'ISI et un égaliseur référencés respectivement **ISI_S** et **Equ** sur la figure. À noter, la présence d'une matrice d'interconnexion reconfigurable, comme présentée par la figure 2.18, permettant d'effectuer le court-circuitage ou la connexion de l'opérateur égaliseur. Le court circuitage permet de réduire à la fois la consommation en arrêtant le fonctionnant de l'opérateur court-circuité ainsi que le temps de traitement. Cette matrice d'interconnexion est contrôlée par L2_ReMU func_ISI.

Nous allons maintenant détailler les différentes phases d'exécution de ce scénario. Il a été défini qu'aucune reconfguration ne pourra être effectuée sur l'opérateur ISI_S, L3_ReMU ISI_S n'intervient donc pas dans le déroulement de ce scénario.

L3_CRMU ISI_S

Dans un premier temps l'opérateur ISI_S envoie une métrique à son L3_CRMU associé relevant la valeur de l'ISI. Celui-ci en fait l'acquisition et traite la métrique de la façon illustrée par la figure 4.5.

Il est à noter que dans l'état actuel d'avancement de HDCRAM (qui est définie pour l'instant de manière purement fonctionnelle) aucune notion de fréquence de relevé des métriques n'est spécifiée. Ceci constituera une piste d'étude pour les futurs travaux liés

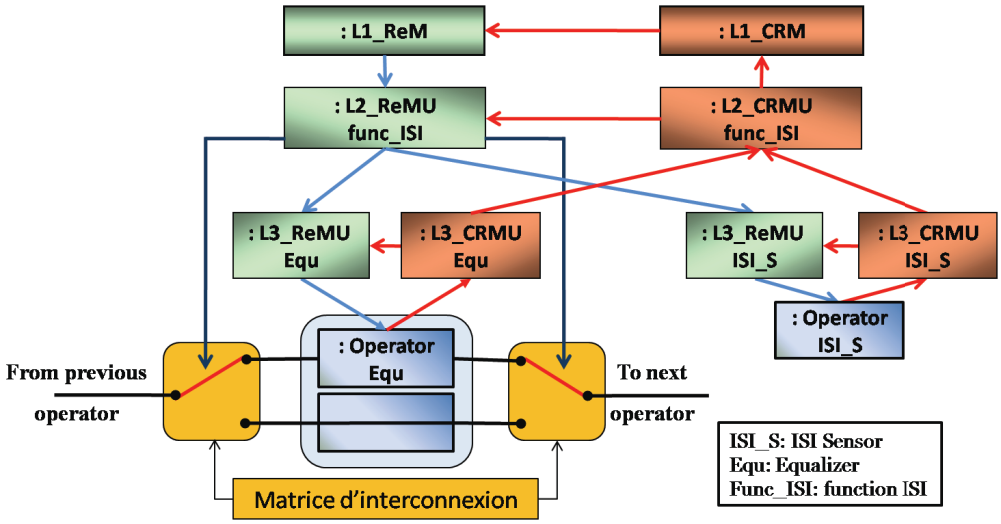


FIG. 4.4: *Architecture HDCRAM utile au scénario ISI*

à HDCRAM. Dans ce scénario, un seuil est attaché à la valeur de la métrique, soit M la métrique :

- Si $M \geq 10\%$ alors M est interprété par L3_CRMU ISI_S par la valeur : “Bad”,
- Si M < 10% alors M est interprété par L3_CRMU ISI_S par la valeur : “Good”,
- Si M = 0% ou 100 % alors il y a probablement un problème dans l'aquisition de la métrique, L3_CRMU ISI_S attend une nouvelle métrique.

Une fois cette interprétation réalisée, et n'ayant pas de reconfiguration à effectuer sur le capteur lui même, la nouvelle métrique est transmise au gestionnaire de l'intelligence de niveau supérieur L2_CRMU func_ISI par L3_CRMU ISI_S.

L2_CRMU func_ISI

L3_CRMU ISI_S transmet la métrique M interprétée à son gestionnaire responsable : L2_CRMU func_ISI. Celui-ci réalise l'acquisition de la métrique et la traite comme illustré par la figure 4.6.

Comme défini dans le fonctionnement, suivant la valeur relevée, cette unité de gestion de l'intelligence va effectuer un choix de reconfiguration. Cette liberté de reconfiguration à été mise en place ici afin de diminuer le temps de réaction de la boucle de décision et ainsi minimiser le temps de reconfiguration. Dans un premier temps, une évaluation de l'état actuel va être effectuée : l'opérateur égaliseur est-il ou non court-circuité ?

Cette information est accessible au travers de la bibliothèque associée contenant les attributs “status” des gestionnaires contrôlés par L2_ReMU func_ISI, notamment le sta-

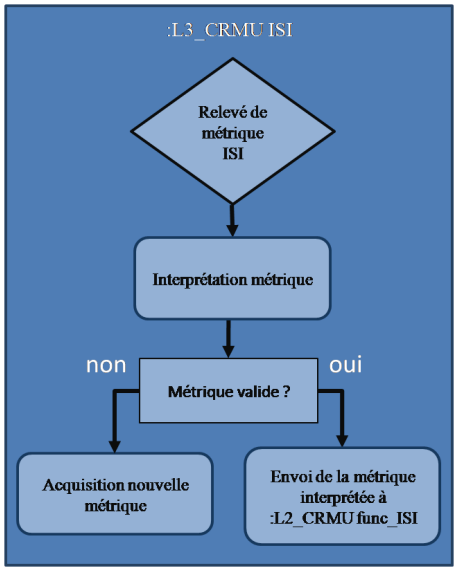


FIG. 4.5: *Interprétation par L3_CRMU ISI_S de la métrique issue de l'opérateur ISI_S*

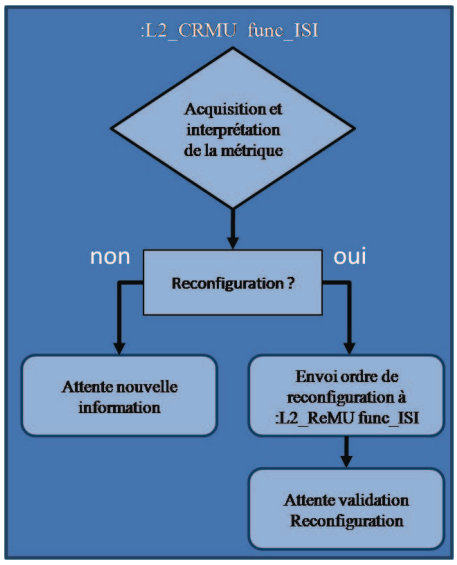


FIG. 4.6: *Interprétation par L2_CRMU func_ISI de la métrique issue de L3_CRMU ISI_S*

tus de L3_ReMU Equ qui relève l'état de fonctionnement de l'opérateur égaliseur dont il a la charge. Suivant l'état de l'opérateur une prise de décision de reconfiguration va être décidée ou non. En effet l'interprétation réalisée au L2_CRMU func_ISI est la suivante :

- Si M = "Bad" alors le canal est perturbé et l'égaliseur est nécessaire
- Si M = "Good" alors le canal est dit favorable et l'égaliseur n'est pas nécessaire

La reconfiguration intervient alors si un changement est à appliquer sur l'égaliseur. Si une reconfiguration est nécessaire, l'ordre est envoyé au L2_ReMU func_ISI de court-circuiter ou de réintégrer l'égaliseur au traitement du signal.

L2_ReMU func_ISI

La figure 4.7 illustre l'acquisition par L2_ReMU func_ISI de l'ordre de reconfiguration émis par L2_CRMU func_ISI ainsi que le traitement de cet ordre.

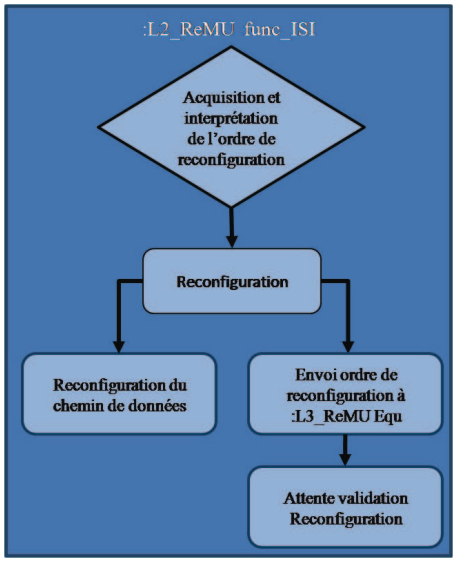


FIG. 4.7: *Aquisition par L2_ReMU func_ISI de l'ordre de reconfiguration issue de L2_CRMU func_ISI*

L'ordre de reconfiguration reçu va tout d'abord être interprété. En effet, le gestionnaire CRMU n'a que faire du comment la reconfiguration est appliquée, il faut juste la faire. Ainsi, l'ordre envoyé ne contient que des renseignement sur le changement d'état d'un l'opérateur : *do not need equalization* ou *need equalization*. L'interprétation permet au L2_ReMU func_ISI de gérer cet ordre de façon à l'appliquer au mieux. Ainsi, après interprétation, il en découle que le L2_ReMU va avoir deux actions à mener et que ces actions peuvent être menées en parallèle. Ces actions sont :

- envoi d'un ordre de reconfiguration au L3_ReMU equalizer pour l'arrêt ou l'activation de l'opérateur de traitement,
- reconfiguration du chemin de données pour court-circuiter l'élément.

Bien entendu, la reconfiguration du chemin de données est également ordonnancée de façon à ne pas perturber le traitement du signal. L'ordre envoyé à L3_ReMU equalizer contiendra le message :

- "stop" si la reconfiguration à effectuer est l'arrêt de l'opérateur,
- "activate" si la reconfiguration à effectuer est la réintégration de l'opérateur au traitement du signal.

L3_ReMU Equ

La figure 4.8 présente les actions du L3_ReMU equalizer une fois l'ordre de son L2_ReMU reçu.

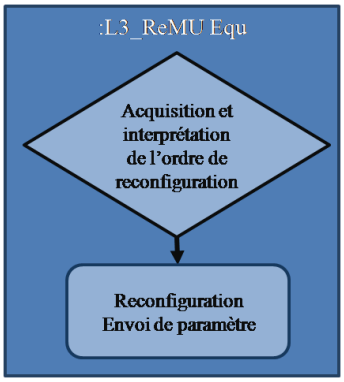


FIG. 4.8: *Acquisition par L3_ReMU Equ de l'ordre de reconfiguration issue de L2_ReMU func_ISI*

L'interprétation qui est faite par cette unité de gestion de reconfiguration est la suivante. Soit O l'ordre reçu :

- Si O = "deactivate" alors envoi d'un signal stoppant l'activité de l'opérateur (arrêt de l'horloge par exemple),
- Si O = "activate" alors envoi d'un signal activant l'opérateur.

Après avoir présenté le principe de ce scénario nous allons introduire les résultats fournis par l'exécution dans l'environnement de travail.

Résultats

L'architecture déployée pour ce scénario a déjà été présentée par la figure 4.3 en introduction comme support à la présentation de l'environnement de travail. Nous allons ici nous intéresser aux modifications visibles dans les attributs après l'exécution d'une simulation. Nous présenterons tout d'abord l'évolution des attributs en partant du niveau de hiérarchie le plus bas pour remonter jusqu'au L1. Les résultats seront présentés suivant le mode illustré par la figure 4.9. Afin de préciser les évolutions de valeurs d'attributs entre deux simulations celles-ci seront mises en valeur par un encadrement.

Property	Value	Value
Properties of attributes	Value at initialization	Value after a run

FIG. 4.9: *Mode de présentation des résultats*

La figure 4.10 présente le module permettant de calculer la valeur de l'ISI et d'interpréter cette valeur.

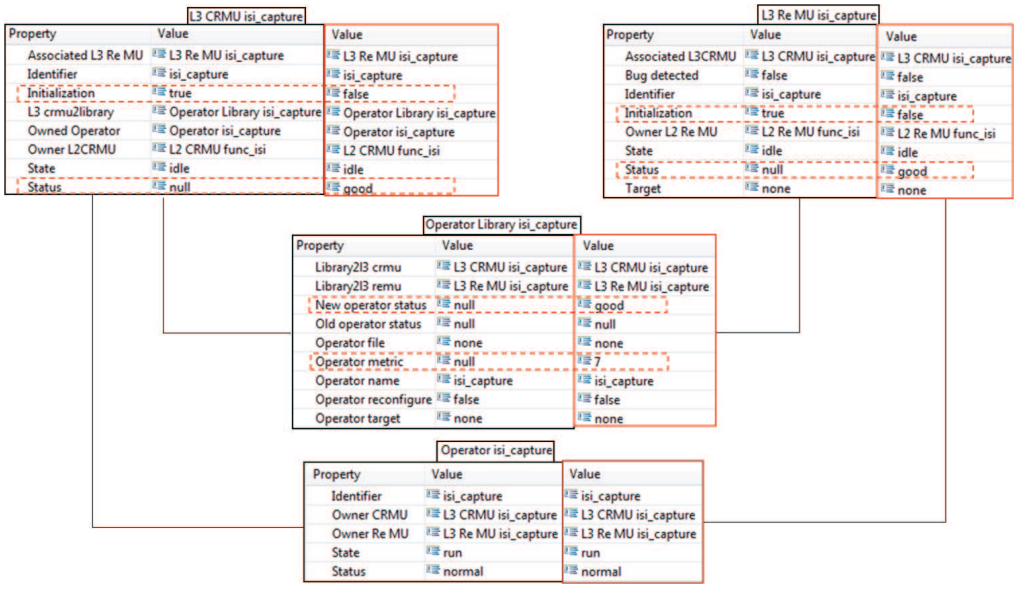


FIG. 4.10: *Module de capture de l'ISI*

Le relevé permettant l'évolution du système est la valeur d'ISI (rentrée par l'utilisateur) référencée dans *Operator Library isi_capture* par la valeur de l'attribut *Operator metric*. Comme mentionné précédemment, la valeur de cette métrique étant inférieure à 10, l'effet final en termes de reconfiguration va être l'arrêt de l'opérateur d'égalisation ainsi que la reconfiguration de la matrice d'interconnexion. Le premier effet de cette métrique est de faire basculer dans le *L3_CRMU isi_capture* la valeur du status, ainsi,

la résultante est l'interprétation de la métrique "7" par une nouvelle métrique "good". L'unité *L3_CRMU equalizer* ne participant pas à l'envoi de métrique pour le déroulement de ce scénario, nous allons maintenant nous intéresser aux répercussion de l'évolution constatée au travers la figure 4.11 représentant le module L2.

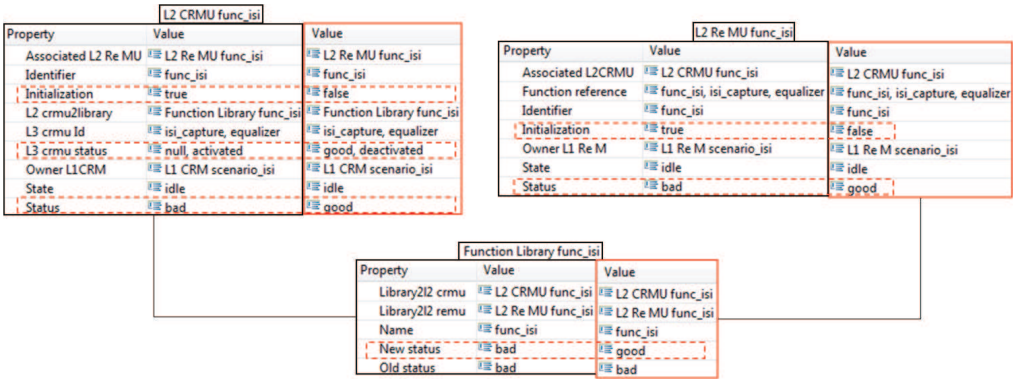


FIG. 4.11: *Module L2*

Le module L2 nous permet de visualiser l'évolution relative à la capture et à l'interprétation de la métrique du *L3_CRMU isi_capture*. On peut voir que les valeurs de l'attribut *L3_crmu_Status* ont été modifiées. La métrique fournie par le gestionnaire de l'intelligence de hiérarchie inférieure est interprétée et permet de prendre une décision de reconfiguration sur le module d'égalisation. Cette valeur étant "good" la décision résultante est de désactiver le module d'égalisation. Cette phase n'est pas représentée par la vue arborescente mais est clairement accessible au travers de l'interface textuelle comme présenté par 4.12.

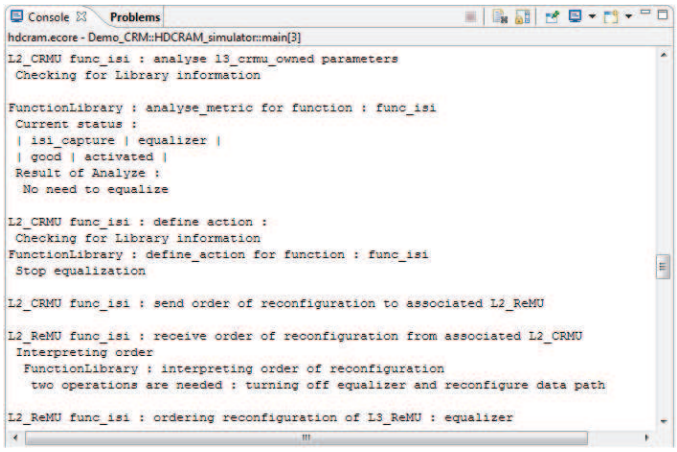


FIG. 4.12: *Interface textuelle - interprétation des attributs fournis par les L3_CRMU afin de prendre une décision de reconfiguration*

On constate que *L2_CRMU func_ISI* interprête le changement de valeur de la métrique *isi_capture* par la suppression de l'égalisation. Il envoie donc à son L2_ReMU

associé, l'ordre "stop equalization" qui est interprété comme nécessitant deux actions : l'arrêt de l'égalisation ainsi que la reconfiguration du chemin de données. L'envoi de l'ordre de reconfiguration implique que *L2_CRMU func_ISI* identifie le gestionnaire de reconfiguration de hiérarchie inférieure en charge de l'égalisation parmi ceux qu'il contrôle. La figure 4.13 présente le module d'égalisation.

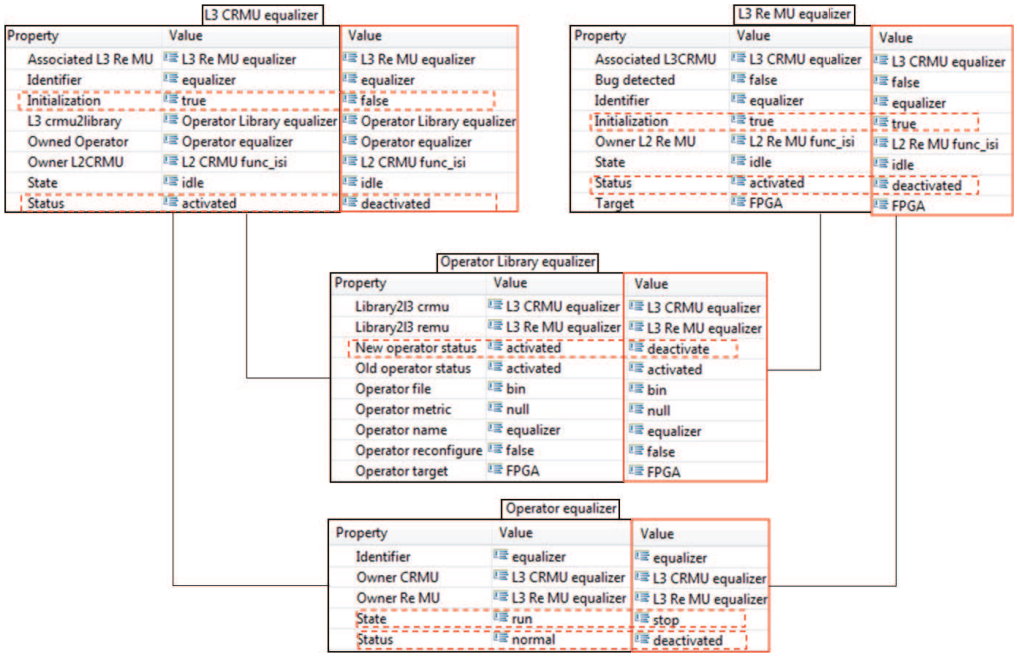


FIG. 4.13: *Module d'égalisation*

Sur cette figure apparaît l'interprétation de l'odre de reconfiguration envoyé par le gestionnaire de reconfiguration de hiérarchie supérieure au travers la valeur de la métrique Status de *L3_ReMU equalizer*. Cette interprétation est illustrée par la séquence présentée par la figure 4.14.

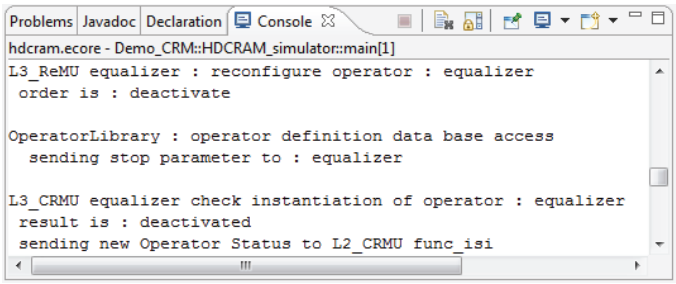


FIG. 4.14: *Interprétation de l'ordre de reconfiguration par L3_ReMU et prise en compte de la validité de la reconfiguration par L3_CRMU*

Après avoir étudié l'évolution des modules L3 et L2, nous allons maintenant étudier les répercussion de l'évolution de la métrique ISI sur le module L1 comme présenté par la figure 4.15.

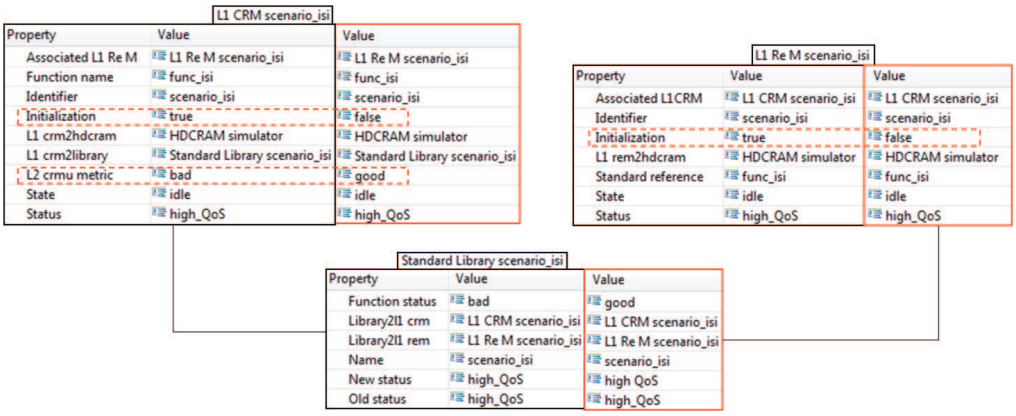


FIG. 4.15: *Module L1*

On observe que les changements relatifs à l'adaptation de la variation de la métrique ISI n'influencent pas le module L1. Les décisions de reconfiguration ont été décidées par un module L2 et n'ont donc pas perturbé le fonctionnement du L1. Nous sommes donc bien dans le cas ou des agents logiciels ont la possibilité de prendre des décisions de reconfiguration sans avoir à en référer à une entité centrale, évitant par la même le risque de goulot d'étranglement et accélérant la prise de décision pour une adaptation plus rapide aux variations de l'environnement.

Résumé

Nous avons montré à travers ce premier scénario une possibilité de reconfiguration tirant avantageusement parti de la distribution hiérarchique des unités de gestion de l'intelligence. Par leurs degrés de liberté, ces gestionnaires peuvent, à l'instar des agents logiciels, opérer des reconfigurations sans en référer à une unité centrale. Ce type de traitement, bien maîtrisé, permet d'éviter des goulots d'étranglement et permet aussi une traitement et donc une réaction plus rapide à une variation. Il est bien entendu que ce type de prise de décision ne faisant intervenir qu'un seul module L2 peut être mis en place pour des variations ayant un impact très localisé sur le système. Néanmoins, la remontée d'information jusque dans la bibliothèque accessible au module L1 permet une prise en compte des cas de figure amenant à une reconfiguration à plus grande granularité.

Afin de permettre à un équipement d'intégrer les fonctionnalités introduites par ce scénario, il lui faut disposer des éléments présentés dans les deux tableaux 4.1 et 4.2 suivants.

TAB. 4.1: *récapitulatif des éléments CRMU intervenant dans l'exécution du scénario*

Unité	Opération	Descriptif
L3_CRMU isi_capture	capture_metric	Permet l'acquisition de la métrique en provenance de l'opérateur isi_capture
	analyze_metric	Interprète la valeur de la métrique : good/bad/ErrorMetric
	define_action	Définit l'action à réaliser • Si ErrorMetric alors faire nouvelle acquisition • Sinon transmettre à L2_CRMU func_isi
L2_CRMU func_isi	capture_metric	Permet l'acquisition de la métrique en provenance de L3_CRMU isi_capture
	analyze_metric	Interprète la valeur de la métrique : • Si bad : need to equalize • Si good : no need to equalize
	define_action	Suivant l'état courant décide d'une reconfiguraton
	send_order2L2_ReMU(order : abstract_order)	Envoi de l'ordre abstrait de reconfiguration au L2_ReMU associé

TAB. 4.2: *récapitulatif des éléments ReMU intervenant dans l'exécution du scénario*

Unité	Opération	Descriptif
L2_ReMU func_isi	receiveL2_CRMUorder(order : abstract_order)	Permet l'acquisition de l'ordre abstrait de reconfiguration du L2_ReMU associé
	interpreteOrder(order : abstract_order)	Interprète l'ordre de reconfiguration : • Reconfigurer le chemin de données • Si need to equalize activer le module égalisation sinon le désactiver
	reconfigureL3_ReMU(target : crm_reference, order : reconfiguration_order)	Cible le L3_ReMU en charge du module égalisation et lui transmet l'ordre de reconfiguration : activate / deactivate
L3_ReMU equalizer	reconfigure	• Reçoit l'ordre de reconfiguration • Ordonnance l'arrêt de l'opérateur à charge
	data_base_access	Interprète l'ordre de reconfiguration : • Envoi de paramètre de reconfiguration • Chargement d'un fichier de configuration

4.4 Scénario2 : Adaptation aux variations de niveau de batterie

Comme déjà évoqué, nous pensons que la RC aura des répercussions qui iront au delà de l'unique gestion de la ressource spectrale. Notamment, la RC permettra d'adapter le fonctionnement des équipements de manière plus judicieuse en fonction des variations des caractéristiques opérationnelles propres de la plateforme d'exécution. L'un de ces enjeux est la gestion de la consommation. Nous allons introduire ici un scénario illustrant l'adaptation du fonctionnement de l'équipement à une variation de niveau de batterie. Pour cela, nous avons pris exemple sur les travaux de [108] qui préconise l'utilisation d'une méthode nommée FPLA (*Functional Level Power Analysis*) afin de modéliser la consommation d'IP sur cible FPGA. L'utilisation de cette méthode permet de modéliser les consommations de diverses IP, il est aussi possible d'utiliser différentes conceptions pour une même IP permettant ainsi d'obtenir une IP déclinée en plusieurs versions suivant sa consommation (ce qui à un effet direct sur ses performances). C'est ce que nous allons utilisé dans ce scénario. Bien que cette méthode, mettant à disposition plusieurs versions d'une même IP pour permettre une adaptation suivant les règles de consommation, puisse être appliquée à l'ensemble des IP de traitement du signal, pour des raisons de clareté, nous ne présenterons ici que l'utilisation sur un seul opérateur.

Soit un opérateur Operator_1 utilisé dans le traitement du signal radio dont il existe différentes versions. On entend, pour ce scénario, par différentes versions, 3 fichiers binaires pour l'implantation de l'opérateur sur cible DSP (faible consommation, consommation moyenne, forte consommation) ainsi que 3 fichiers bitstreams pour implantation sur FPGA. L'utilisation de l'une ou l'autre des versions est soumise à la possibilité d'amoindrir la QoS du signal audio ou video (dans une limite acceptable du point de vue de l'utilisateur) afin de diminuer la consommation pour prolonger le maintient de la communication. En effet nous admettrons que pour une même IP la performance visée influe généralement sur la consommation. La figure 4.16 présente l'architecture déployée pour l'exécution de ce scénario.

À la différence du scénario 1, la reconfiguration qui sera effectuée ici sera le chargement sur cible d'un nouveau fichier de configuration et non un simple envoi de paramètres. Nous ferons ici l'hypothèse que la place (en terme de portes si la cible est un FPGA par exemple) restante sur la cible n'est pas une variable à prendre en considération dans le choix de l'IP à implanter et que seul le niveau de la batterie est important. Ainsi, à un niveau bas de batterie correspond le chargement du fichier étiquetté *basse performance*, à un niveau moyen de batterie correspond le fichier *moyenne performance* et enfin à un niveau haut de batterie correspond le chargement du fichier étiquetté *haute performance*.

Dans l'exécution de ce scénario, l'utilisateur a la possibilité d'effectuer un changement de cible technologique pour l'implantation de l'opérateur lors de la reconfiguration.

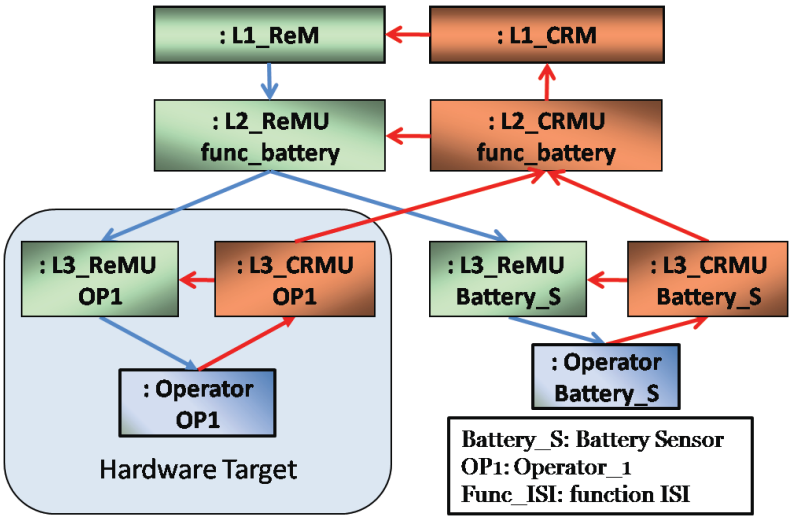


FIG. 4.16: *Architecture déployée pour l'exécution du scénario 2*

Ceci peut permettre de réduire davantage la consommation et c'est une liberté qui peut être laissée à HDCRAM. Lors d'une telle reconfiguration deux choix peuvent être effectués comme une reconfguration à la volée ou hors ligne. Dans le premier cas, l'architecture doit permettre la mise en place de l'implantation de l'opérateur sur la nouvelle cible en parallèle de l'exécution de ce même opérateur sur l'ancienne cible afin de ne pas interrompre le traitement. Une fois l'implantation effective, une reconfiguration du chemin de données sera effectuée afin de basculer le traitement sur la nouvelle cible.

Nous allons maintenant détailler les différentes phases d'exécution. Il a été défini qu'aucune reconfguration ne pourra être effectuée sur l'opérateur battery_sensor, l'unité L3_ReMU battery_sensor n'intervient donc pas dans le déroulement de ce scénario.

L3_CRMU battery_sensor

Dans un premier temps l'opérateur battery_sensor envoie une métrique à son unité L3_CRMU associée relevant le niveau de la batterie. Celui-ci en fait l'acquisition et traite la métrique de la façon illustrée par la figure 4.17.

Dans ce scénario, un seuil est attaché à la valeur de la métrique, soit M la métrique :

- Si $M > 0\%$ et M < 31% alors M est interprété par L3_CRMU battery_sensor par la valeur : "low",
- Si $M > 30\%$ et M < 71 alors M est interprété par L3_CRMU battery_sensor par la valeur : "moderate",
- Si $M > 70\%$ et $M \leq 100$ alors M est interprété par L3_CRMU battery_sensor par la valeur : "high",

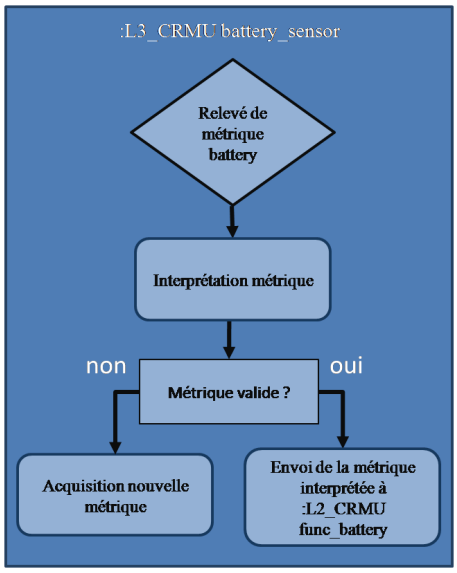


FIG. 4.17: *Interprétation par L3_CRMU battery_sensor de la métrique issue de l'opérateur battery_sensor*

- Si $M \leq 0\%$ ou si M > 100% alors il y a un problème dans l'aquisition de la métrique, L3_CRMU battery_sensor attend une nouvelle métrique.

Une fois cette interprétation réalisée, et n'ayant pas de reconfiguration à effectuer sur l'opérateur dont il a la charge, la nouvelle métrique est transmise au gestionnaire de l'intelligence de niveau supérieur L2_CRMU func$_batterypar L3_CRMU battery_sensor$.

L2_CRMU func_battery

L3_CRMU battery_sensor transmet la métrique M interprétée à son gestionnaire responsable : L2_CRMU func_battery. Celui-ci réalise l'acquisition de la métrique et la traite comme illustré par la figure 4.18.

Suivant la valeur relevée, cette unité de gestion de l'intelligence va effectuer un choix de reconfiguration. Dans un premier temps, l'unité va vérifier si il y a une demande de changement de cible technologique.

Changement de cible technologique

Dans HDCRAM, cela se traduit par la création d'un nouveau module L3 restant sous le contrôle du module L2 responsable de l'ancien module L3. En effet, bien que technologiquement et physiquement les contraintes ne soient plus les mêmes, fonctionnellement,

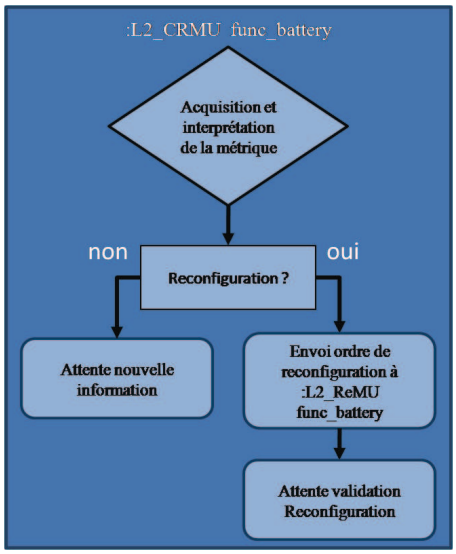


FIG. 4.18: *Interprétation par L2_CRMU func_battery de la métrique issue de L3_CRMU battery_sensor*

le traitement reste identique. La figure 4.19 illustre le séquençage des opérations pour un traitement de reconfiguration de type à la volée. Il est à noter que cette figure n'est fournit ici qu'à titre d'illustration et n'intervient pas dans le flot de conception.

Sur cette figure 4.19, on remarque que la destruction du module A (L3_CRMU A, L3_ReMU A, Operator A) est ordonnancée après la réussite de l'implantation du module B (qui assure les mêmes fonctions que le module A sur une nouvelle cible technologique) ainsi qu'après la réussite de la reconfiguration du chemin de données. Si l'ordre arrivant à un L2_ReMU est la création d'un nouveau module L3 voici le séquençage des opérations :

- le L2_ReMU crée le nouveau L3_ReMU,
- Le L3_ReMU invoque son L3_CRMU associé,
- Le L3_ReMU implante l'opérateur dont il a la charge.

Si l'ordre arrivant à un L2_ReMU est la destruction d'un module L3 voici le séquençage des opérations :

- le L2_ReMU envoie l'ordre de destruction au L3_ReMU,
- Le L3_ReMU détruit l'opérateur dont il a la charge,
- Le L3_ReMU détruit son L3_CRMU associé,
- le L2_ReMU détruit L3_ReMU.

La figure 4.20 quant à elle, illustre un traitement de reconfiguration de type hors ligne.

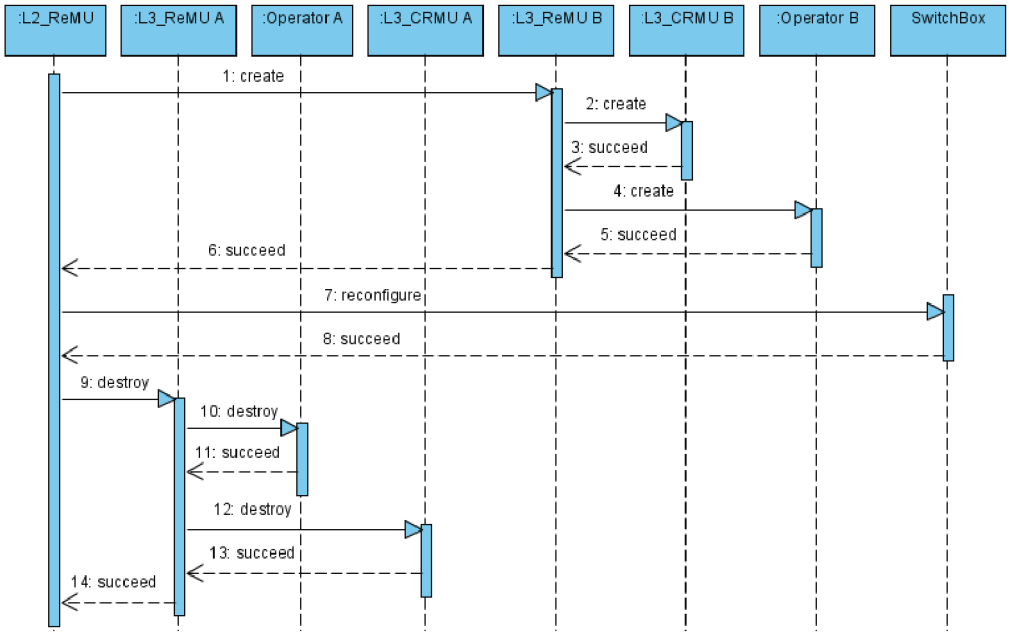


FIG. 4.19: *Réallocation d'un opérateur à la volée*

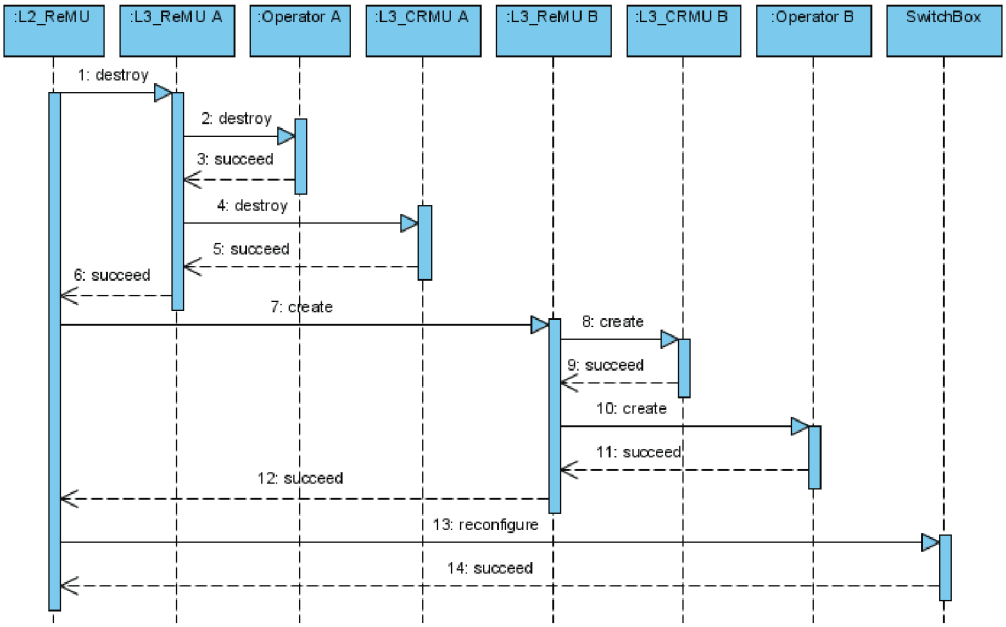


FIG. 4.20: *Réallocation d'un opérateur hors ligne*

En effet, pour ce deuxième cas où l'on peut se permettre une reconfiguration hors ligne il n'y a pas nécessité de garder l'ancien opérateur en fonctionnement jusqu'à ce que la dérivation soit mise en place. Le séquençage des opérations de création et de destruction des différents éléments est identique au cas de reconfiguration à la volée, seules changent les priorités de reconfigurations gérées par le L2_ReMU responsable de la réallocation.

Non changement de cible technologique

S'il n'y a pas de changement de cible technologique, alors une évaluation de l'état actuel va être effectuée : quel est le status de l'opérateur implanté ?

Cette information est accessible au travers de la bibliothèque associée contenant les attributs "status" des gestionnaires contrôlés par L2_ReMU func_battery, notamment le status de L3_ReMU Operator_1 qui relève l'état de fonctionnement de l'opérateur dont il a la charge.

Suivant l'état de l'opérateur une prise de décision de reconfiguration va être décidée ou non. En effet l'interprétation réalisée au L2_CRMU func_ISI est la suivante :

- Si M = "low" alors le niveau de la batterie est bas et il faut tout faire pour économiser de l'énergie,
- Si M = "moderate" alors le niveau de batterie est moyen et ne nécessite pas de mesure d'économie d'énergie,
- Si M = "high" alors le niveau de batterie est élevé et il est possible de faire fonctionner le système à pleine charge.

La reconfiguration intervient alors si un changement est à appliquer sur l'opérateur_1. Si une reconfiguration est nécessaire, l'ordre est envoyé au L2_ReMU func_battery de modifier la performance de de la fonction réalisée par Operator_1.

L'ordre envoyé à L2_ReMU associé contiendra le message :

- "decrease performance" si la reconfiguration doit à tous prix permettre d'économiser de l'énergie,
- "moderate performance" si la reconfiguration à effectuer est la recherche d'une performance moyenne (compromis consommation/performance),
- "increase performance" si la reconfiguration permet d'utiliser les ressources au maximum de leurs possibilités.

L2_ReMU func_battery

La figure 4.21 illustre l'acquisition par L2_ReMU func_battery de l'ordre de reconfiguration émis par L2_CRMU func_battery ainsi que le traitement de cet ordre.

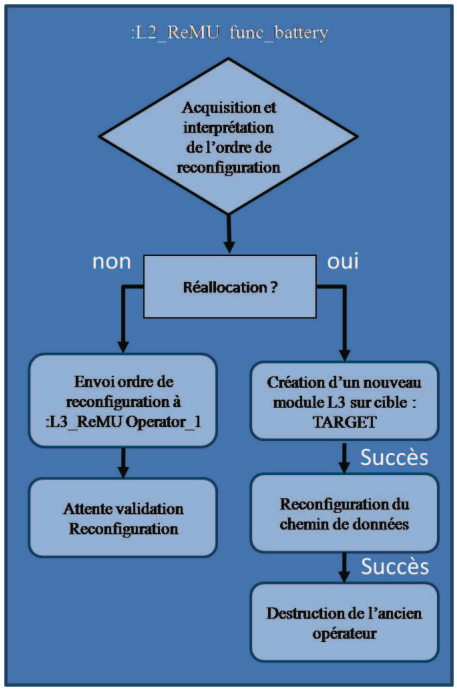


FIG. 4.21: *Acquisition par L2_ReMU func_battery de l'ordre de reconfiguration issu de L2_CRMU func_battery*

L'ordre de reconfiguration reçu va tout d'abord être interprété, en effet, le gestionnaire CRMU n'a que faire de la manière dont la reconfiguration est appliquée, il faut juste la faire. Ainsi, l'ordre envoyé ne contient que des renseignements sur le changement d'état de l'opérateur : *decrease performance* ou *increase performance*.

L'interprétation qui est faite par cette unité de gestion de reconfiguration est la suivante. Soit O l'ordre reçu :

- Si O = "decrease performance" alors si l'état courant n'est pas déja en mode économie d'énergie, le faire.
- Si O = "moderate performance" alors
  - si l'état courant est en économie d'énergie, améliorer sa performance (choix de performance modérée)
  - si l'état courant est en meilleure performance, la diminuer.
- Si O = "increase performance" alors si l'état courant n'est pas déja en mode meilleure performance, le faire.

L'ordre envoyé à L3_ReMU Operator_1 contiendra le message :

- “perf_low” si la reconfiguration doit à tous prix permettre d’économiser de l’énergie,
- “perf_moderate” si la reconfiguration à effectuer est la recherche d’une performance moyenne (compromis consommation/performance),
- “perf_high” si la reconfiguration permet d’utiliser les ressources au maximum de leurs possibilités.

À cet ordre est juxtaposé l’identifiant de cible matérielle : _DSP ou _FPGA.

L3_ReMU Operator_1

La figure 4.22 présente les actions du L3_ReMU Operator_1 une fois l’ordre de son L2_ReMU reçu.

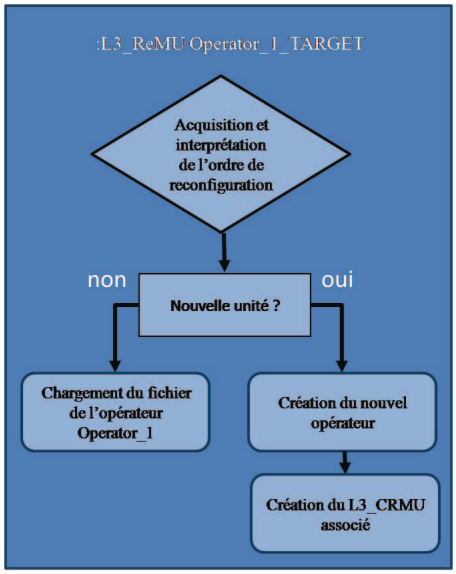


FIG. 4.22: *Acquisition par L3_ReMU Operator_1 de l’ordre de reconfiguration issu de L2_ReMU func_battery*

L’interprétation qui est faite par cette unité de gestion de reconfiguration est la suivante, soit O l’ordre reçu :

- Si O = “perf_low” alors chargement du fichier Operator_1_perf_low.
- Si O = “perf_moderate” alors chargement du fichier Operator_1_perf_moderate.
- Si O = “perf_low” alors chargement du fichier Operator_1_perf_high.

À cet appel de fichier est juxtaposé l’identifiant de cible matérielle : .bin si la cible est un FPGA ou .exe si la cible est un DSP.

Après avoir présenté le principe de ce scénario nous allons introduire les résultats fournis par l'exécution dans l'environnement de simulation.

Résultats

Nous présenterons tout d'abord l'évolution des attributs en partant du niveau de hiérarchie le plus bas pour remonter jusqu'au L1. Les résultats seront présentés suivant le mode illustré par la figure 4.23. Afin de préciser les évolutions de valeurs d'attributs entre deux simulations celles-ci seront mises en valeur par un encadrement.

Property	Value	Value	Value
Properties of attributes	Value at initialization	Value at intermediate time	Value after a run

FIG. 4.23: *Mode de présentation des résultats*

L'architecture déployée pour ce scénario est présentée par la figure 4.24.

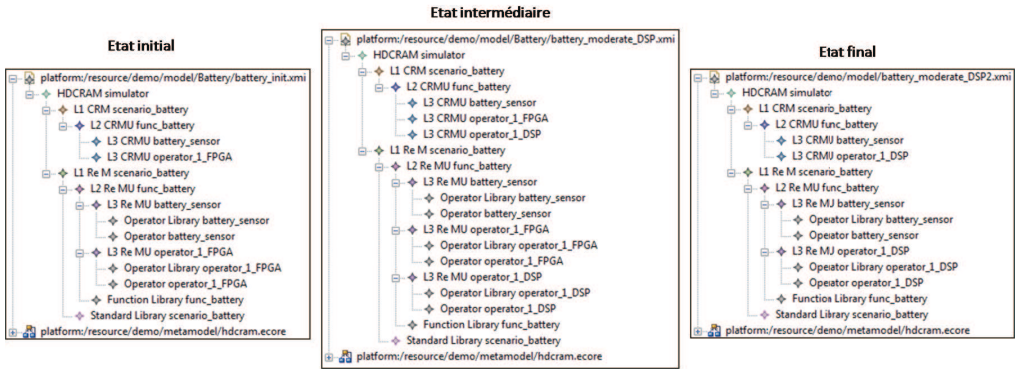


FIG. 4.24: *Représentation des éléments instanciés*

Pour le déroulement de ce scénario nous avons choisi de montrer comment était pris en compte la décision de changement de cible technologique pour un opérateur lors d'une reconfiguration à la volée. L'opérateur sera transféré d'une cible FPGA vers une cible DSP. L'étape de reconfiguration du chemin de données prise en compte par HDCRAM par l'unité *L2_ReMU func_battery* sera illustrée par la sortie du simulateur. De plus la performance de l'opérateur sera diminuée et tombera du niveau de plus haute performance à un niveau intermédiaire.

Nous allons tout d'abord nous intéresser au module faisant l'acquisition du niveau de batterie. Ce module est présenté par la figure 4.25.

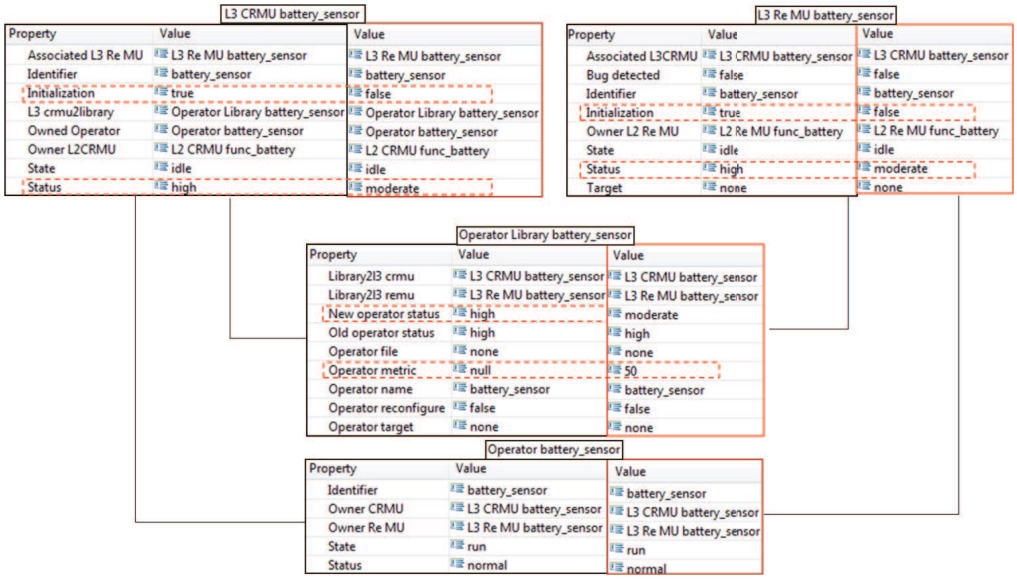


FIG. 4.25: *Vue du module L3 sensor*

Le relevé permettant l'évolution du système est la valeur du niveau de batterie (rentrée actuellement par l'utilisateur) repéré dans *Operator Library battery_sensor* par la valeur de l'attribut *Operator metric*. Comme mentionné précédemment, la valeur de cette métrique étant comprise entre 30 et 70, l'effet final en termes de reconfiguration va être le chargement du fichier de reconfiguration nommé Operator_1_perf_moderate. Le premier effet de cette métrique est de faire basculer dans le *L3_CRMU battery_sensor* la valeur du status, ainsi, la résultante est l'interprétation de la métrique "50" par une nouvelle métrique "moderate". L'unité *L3_CRMU Operator_1* ne participant à l'envoi de métrique pour le déroulement de ce scénario, nous allons maintenant nous intéresser aux répercussion de l'évolution constatée au travers la figure 4.26 représentant le module L2.

Le module L2 nous permet de visualiser l'évolution relative à la capture et à l'interprétation de la métrique du *L3_CRMU battery_sensor*. On peut voir que les valeurs de l'attribut *L3_crmu_Status* ont été modifiées. La métrique fournie par le gestionnaire de l'intelligence de hiérarchie inférieure est interprétée et permet de prendre une décision de reconfiguration sur le module Operator_1.

À cet instant de la simulation, l'utilisateur fait le choix de réallouer le traitement effectué par l'opérateur Operator_1 sur une nouvelle cible technologique (de FPGA vers DSP) comme présenté par la figure 4.27.

Il est à noter que la réponse du système effectuant une recherche des ressources disponibles n'est ici défini que dans un but didactique. Cette information nécessite la mise en place de données, dans la bibliothèque accessible à ce gestionnaire, relatives aux taux d'occupations des différentes cibles technologiques potentielles pour la réallocation de l'opérateur.

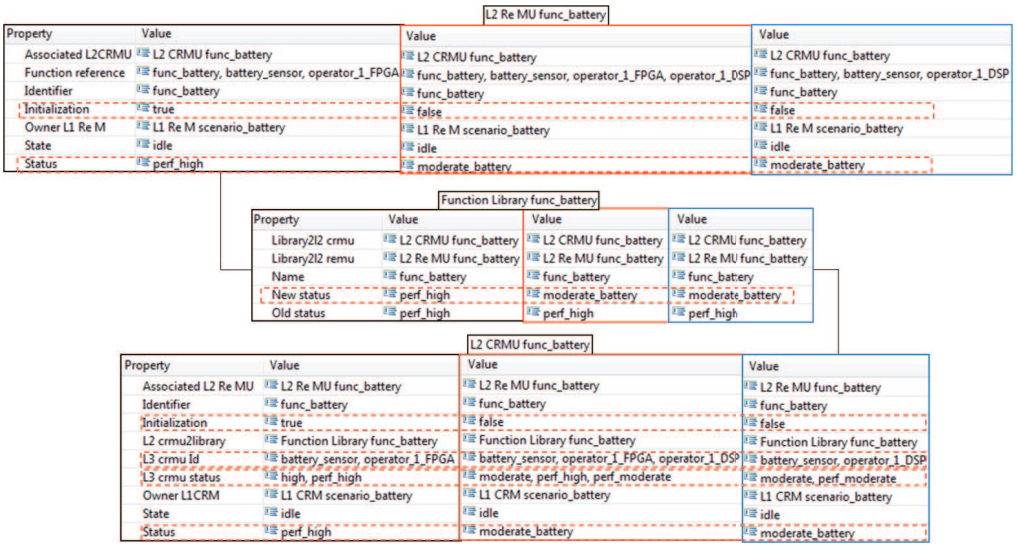


FIG. 4.26: *Vue du module L2*

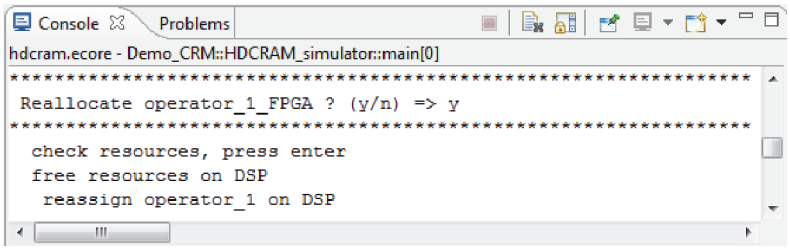


FIG. 4.27: *Interface textuelle - changement de la cible d'exécution matérielle*

La volonté de réallocation exprimée par l'utiliseur déclenche ici une suite d'opération menant à l'implantation de l'opérateur Operator_1 sur une cible DSP avec une performance moyenne (adaptation à la métrique précédemment relevée). Une partie de cette suite d'opération est détaillée sur la figure 4.28 suivante.

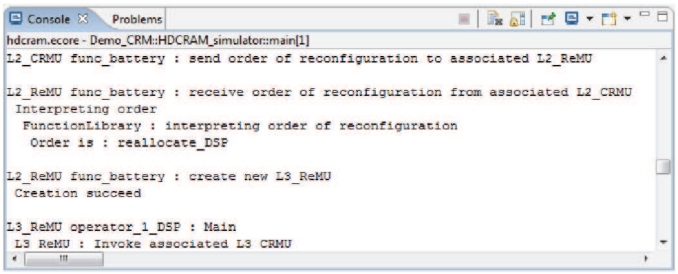


FIG. 4.28: *Interface textuelle - création d'un nouveau L3_ReMU et de son L3_CRMU associé*

On y voit comment le L2_CRMU envoie l'ordre de reconfiguration à son L2_ReMU associé qui interprete l'order "reallocate_DSP" en créant un nouveau L3_ReMU. Ce dernier invoque son L3_CRMU associé en lui transmettant les paramètres de fonctionnement avant de charger le fichier bianire "operator_1_DSP.perf_moderate.exe" comme présenté par la figure 4.29.

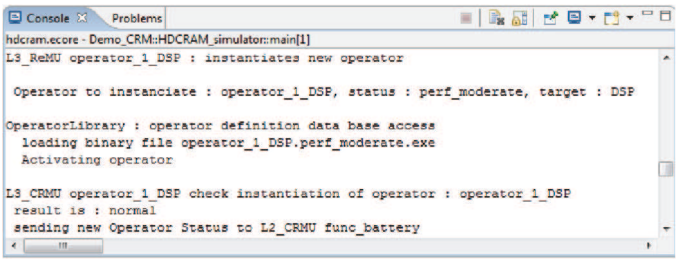


FIG. 4.29: *Interface textuelle - création du nouvel opérateur et vérification de son implantation*

Il apparaît qu'une fois le chargement du fichier de reconfiguration de l'opérateur et donc de son implantation sur la nouvelle cible, le L3_CRMU, qui est alors connecté à cet opérateur, envoie les nouvelles informations à son L2_CRMU associé. Celui ci se voit donc pour l'instant chargé de controler non pas deux mais trois unités de gestion de l'intelligence comme présenté dans l'état intermédiare de la figure 4.26 par la propriété "L3_crmu_Id". La création du nouvel opérateur ayant été un succès, le L2_ReMU va reconfigurer le chemin de données et détruire l'ancien opérateur. Ces opérations sont illustrées par la figure 4.30.

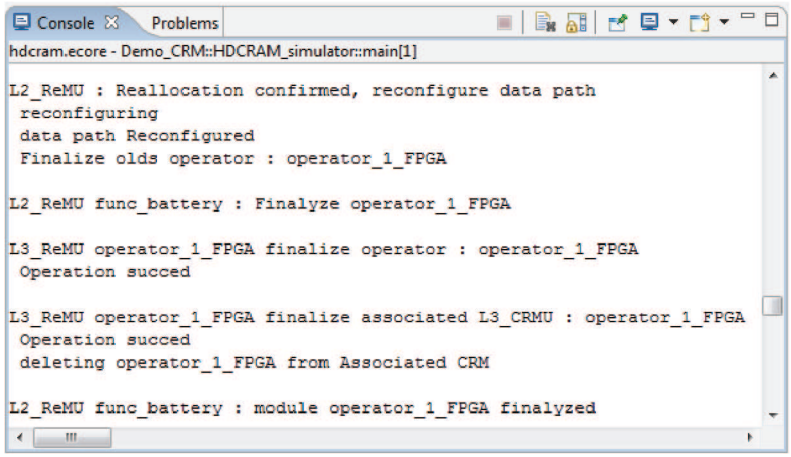


FIG. 4.30: *Reconfiguration du chemin de données et destruction de l'ancien module*

La figure 4.31 présente le nouvau module Operator_1.

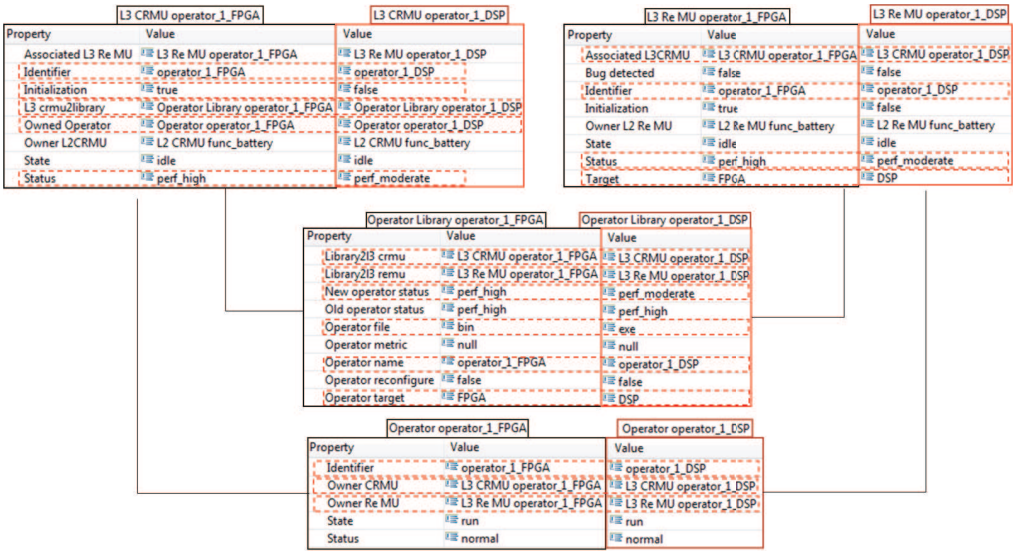


FIG. 4.31: *Module opérateur après réallocation*

Le changement de cible apparaît clairement ainsi que le changement de performance. Après avoir étudié l'évolution des modules L3 et L2, nous allons maintenant étudier les répercussion de l'évolution de la métrique battery sur le module L1 comme présenté par la figure 4.32.

On observe que les changements relatifs à l'adaptation de la variation de la métrique acquis par l'opérateur battery_sensor sont visibles dans le changement de status. En effet l'évolution du niveau de la batterie peut avoir des répercussions sur plusieurs modules, l'entité L1_CRM décidera des possibles modifications à apporter en fonction des paramètres qu'il a à disposition.

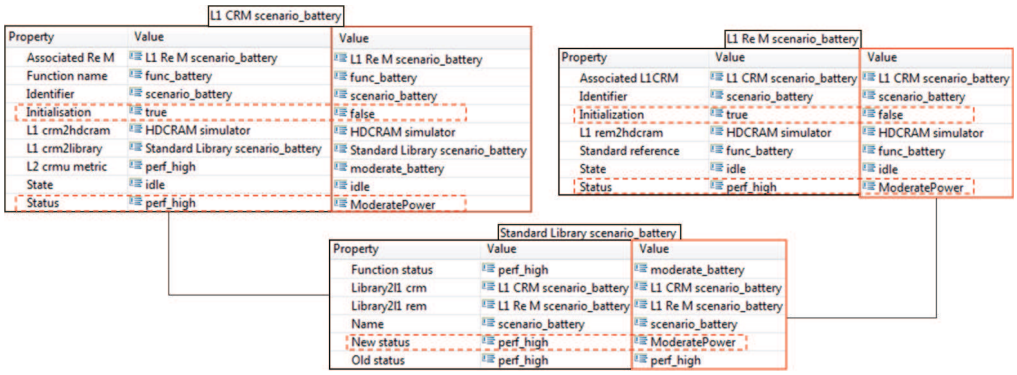


FIG. 4.32: *Vue du module L1*

Résumé

Nous avons illustré dans ce second scénario de quelle manière le système réagit à une variation pouvant entrainer une réallocation d'un opérateur matériel de traitement. Dans ce cas de figure, nous avons montré que bien que les contraintes variaient aussi bien technologiquement que physiquement entre une implantation sur un FPGA ou un DSP par exemple (il pourrait tout autant sagir d'une implantation sur GPP), fonctionnellement, l'utilisation de l'opérateur réalloué restait la même dans la chaîne de traitement. Cela ce traduit dans HDCRAM par le maintien du contrôle du nouveau module L3 par le même module L2. Ce module L2 ayant la charge d'assurer la reconfiguration du chemin de données afin de connecter le nouvel opérateur au reste de la chaîne.

Le résultat de l'utilisation du simulateur, comme outil dans le flot de conception, permet de déuire que afin de permettre à un équipement d'intégrer les fonctionnalités introduites par ce scénario. Ainsi, il lui faut disposer des éléments présentés dans les deux tableaux 4.3 et 4.4 suivants.

TAB. 4.3: *récapitulatif des éléments CRMU intervenant dans l'exécution du scénario*

Unité	Opération	Descriptif
L3_CRMU battery sensor	capture_metric	Permet l'acquisition de la métrique en provenance de l'opérateur battery_sensor
	analyze_metric	Interprète la valeur de la métrique : low/moderate/high/ErrorMetric
	define_action	Définit l'action à réaliser • Si ErrorMetric alors faire nouvelle acquisition • Sinon transmettre à L2_CRMU func_battery
L2_CRMU func battery	capture_metric	Permet l'acquisition de la métrique en provenance de L3_CRMU battery_sensor
	analyze_metric	Interprète la valeur de la métrique : • Si low : low_battery • Si moderate : moderate_battery • Si high : high_battery
	define_action	Suivant l'état courant décide d'une reconfiguraton
	send_order2L2_ReMU(order : abstract_order)	Envoie de l'ordre abstrait de reconfiguration au L2_ReMU associé

TAB. 4.4: *récapitulatif des éléments ReMU intervenant dans l'exécution du scénario*

Unité	Opération	Descriptif
L2_ReMU battery sensor	receiveL2_CRMUorder(order : abstract_order)	Permet l'acquisition de l'ordre abstrait de reconfiguration du L2_ReMU associé
	interpreteOrder(order : abstract_order)	Interprète l'ordre de reconfiguration : • Si reallocate : met en place le changement de cible (création d'un nouveau module Operator_1 et modification du chemin de données)
	reconfigureL3_ReMU(target : crm_reference, order : reconfiguration_order)	Cible le L3_ReMU en charge du module Operator_1 et lui transmet l'ordre de changement de performance
	create_L3_ReMU (NewIdentifier : rem_reference, Newtarget : OperatorTarget, NewStatus : status)	Création d'un nouveau L2_ReMU pour la mise en œuvre du nouveau standard de communication
	finalyzeL3_ReMU(target : L3_ReMU)	Cible et drétruit le L2_ReMU n'étant plus nécessaire (attend la validation par le L2_ReMU de la possibiilté de destruction)
	reconfigureDataPath()	Reconfiguration du chemin de donnée si nécessaire
L3_ReMU Operator_1	reconfigure	• Reçoit l'ordre de reconfiguration • Ordonnance l'arrêt de l'opérateur à charge
	data_base_access	Interprète l'ordre de reconfiguration : • Chargement d'un fichier de configuration
	finalyze()	• Détruit son opérateur associé ainsi que son L3_CRMU associé.

4.5 Scénario3 : accès opportuniste au spectre

Le scénario que nous allons présenter dans cette partie est un cas de reconnaissance de standard en aveugle. C'est-à-dire que l'équipement n'a pas d'information sur les standards utilisables dans son environnement proche et va donc devoir mettre en place une reconnaissance de standard et configurer une chaîne de traitement permettant de satisfaire les besoins de l'utilisateur pour sa communication, son besoin d'accéder à l'internet par exemple ou encore regarder une émission de télévision. C'est un cas futuriste d'accès opportuniste au spectre, qui permet à un équipement d'éviter d'avoir à se connecter à toute une série d'infrastructures afin d'obtenir des informations. Ce qui offre un gain en terme de rapidité et de complexité. La terminologie utilisée pour se scénario diffère des précédents dans le sens ou un module regroupera ici non plus un couple CRM/ReM mais un ensemble de couple L3_ReMU/L3_CRMU/Operator. Ce scénario s'appuie sur un capteur de reconnaissance en aveugle de standard présenté dans [109] et intégré dans notre architecture. Ce capteur, présenté par la figure 4.33, est composé de deux modules principaux, l'un participant à la détection d'énergie dans les bandes RF et l'autre permettant l'analyse du signal détecté.

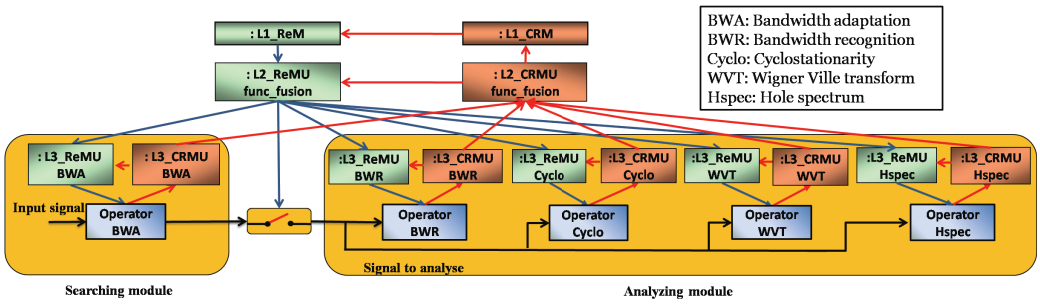


FIG. 4.33: *Ensemble d'éléments participant à la reconnaissance en aveugle du standard*

L'architecture présentée par la figure 4.33 n'intègre que les éléments participant à la détection de standard, en effet, la chaîne de communication est beaucoup plus conséquente. Ceci met en valeur la particularité qu'a notre approche de radio cognitive : elle peut être dimensionnée pour chaque situation, facilitant ainsi une transition progressive vers la radio cognitive. En effet, la surcharge de complexité induite par l'architecture de gestion à ajouter aux organes de traitement du signal radio peut paraître rédhibitoire si elle est appliquée à l'ensemble des éléments de l'équipement. C'est ce à quoi on arrivera à long termes. Cependant, nous sommes persuadés qu'à court termes, c'est pas à pas que la radio cognitive pourra s'imposer, en convainquant peu à peu de son efficacité. On observe dans l'exemple de la figure 4.33 que les deux modules du capteur sont contrôlés par un seul couple L2_ReMU/L2_CRMU, ce choix a été fait de manière à permettre une reconfiguration aisée du chemin de données entre les deux modules.

À la différence des autre scénario présentés ci dessus, nous proposons ici d'illustrer de quelle manière intervient le module L1 pour une prise de décision de reconfiguration.

Il ne s'agit donc pas de reconfiguration locale à granularité moyenne à fine mais bien de reconfiguration pouvant amener à la reconfiguration complète de la chaîne de traitement. Nous allons maintenant détailler les différentes phases d'exécution. Il a été défini qu'aucune reconfguration ne sera effectuée sur les opérateur par leur unité L3_ReMU respective, de fait, ces unités n'interviennent donc pas dans le déroulement de ce scénario et peuvent être réduite à leur plus simple expression.

Le fonctionnement du capteur est le suivant. Tout d'abord, le module de recherche discrimine le signal d'entré par un processus itératif permettant d'isoler les bandes occupées de manière à être analysée par le module d'analyse. Puis ce module d'analyse traite les bandes sélectionnées par le premier module. N'étant pas l'objet de ce travail d'expliquer la mise en œuvre de cela, nous nous contenterons ici d'une description fonctionnelle. Le module d'analyse est composé de quatre opérateurs de traitement comme décrit dans [109] :

- Bandwidth recognition : extrait une métrique en fonction de la ressemblance de la forme du signal reçu avec un ensemble prédéterminé de standards,
- Cyclostationnarity : dicrimine le signal entre signal mono ou multi porteuse,
- Wigner-Ville transform : discrime entre les modes d'étalement soit saut de fréquence (FH) et séquence direct (DS),
- Hole spectrum : détecte les bandes libre de fréquences.

L'exécution de ce scénario induit une demande de l'utilisateur amenant à une recherche d'un standard dans la proximité de l'équipement permettant la satisfaction de son besoin. La figure 4.34 présente sous forme de diagramme d'activité la séquence d'opération relative à la demande jusuqu'à l'obtention d'un standard.

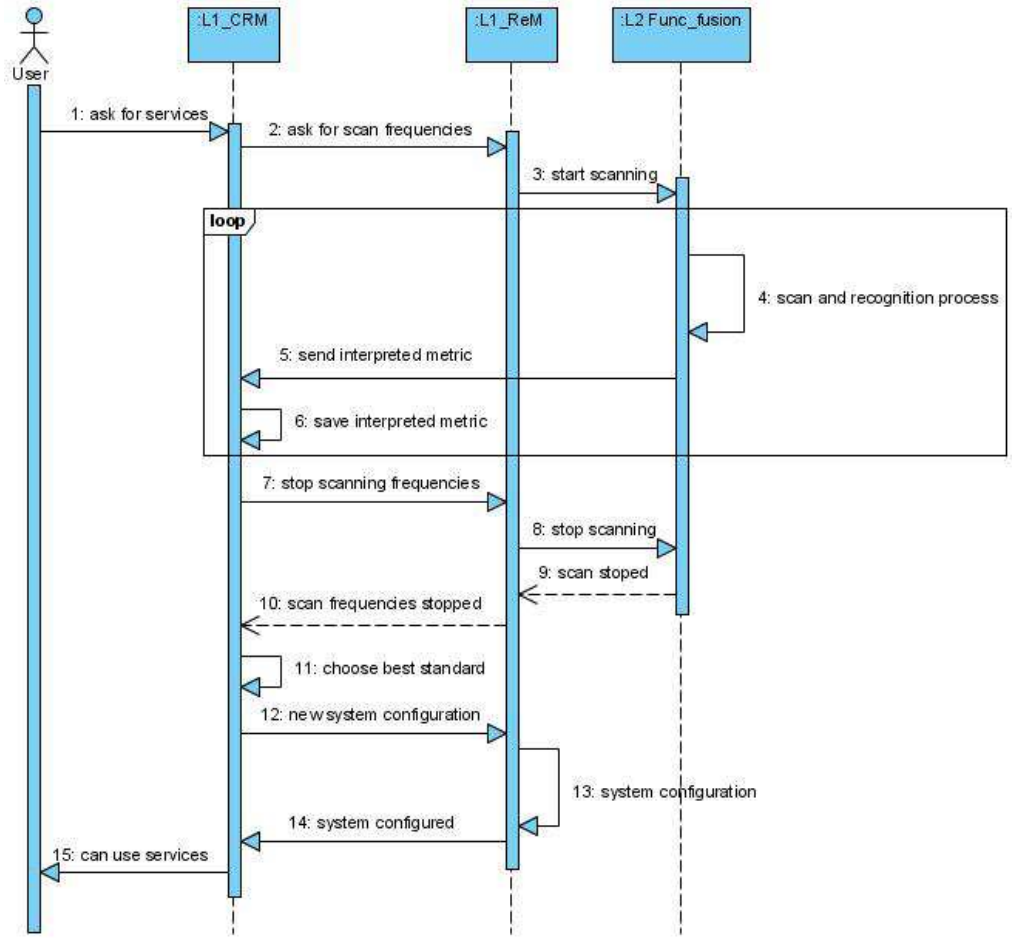


FIG. 4.34: *Phase de recherche de standards utilisables*

Le but du module d'analyse est de fournir au L2_CRMU un ensemble de métriques qui seront validées les unes par rapport aux autres et interprétées avant d'être fournies au L1_CRM. Ce dernier effectuera l'identification du standard, à l'aide de sa bibliothèque associée contenant une table permettant la concordance entre un standard et la métrique fournie par le L2_CRMU comme présenté par la figure 4.5.

TAB. 4.5: *Table de concordance standard - métriques*

Standard	PBc	Channel Filter	FH/DS	Mono/multi
IS-54/IS-136	30 kHz	-	-	Mono
IS-95	1250 kHz	-	-	Mono
DCS 1800/DCS 1900	200 kHz	Gauss 0.3	-	Mono
PDC	25 kHz	-	-	Mono
CT2	100 kHz	Gauss 0.5	-	Mono
GSM	200 kHz	Gauss 0.3	-	Mono
EDGE	200 kHz	Linearized Gauss	-	Mono
GPRS	200 kHz	Gauss 0.3	-	Mono
PHS	300 kHz	Nyquist 0.5	-	Mono
Bluetooth	1 MHz	Gauss 0.5	FH	Mono
WiFi(b)	1 MHz	Gauss 0.5	DS	Mono
IS95	1.25 MHz	FIR	-	Mono
Globalstar	1.25 MHz	RIF48 coeff	-	Mono
DAB	1.712 MHz	Window	-	Multi
DECT	1.728 MHz	Gauss 0.5	-	Mono
UMTS	5 MHz	Nyquist 0.2	-	Mono
DVB-T	7–8 MHz	Window	-	Multi
RLAN	10 MHz	Nyquist	-	Multi
Hiperlan I	20 MHz	-	-	-
LMDS	36 MHz	Nyquist 0.2	-	Mono
Hiperlan II	50 MHz	Window	-	-

On remarque dans ce tableau que généralement la largeur de bande du canal (PBc) du signal discriminé suffit à son identification, néanmoins dans certains cas comme entre bluetooth et le wifi(b), la différence se fait sur le mode de propagation. Il est à noter que plusieurs options peuvent être choisies pour la liberté d'action du choix du standard à utiliser :

- le L1_CRM dispose de tout le temps nécessaire pour recencer l'ensemble des ressources spectrales disponibles et effectuer son choix,
- le L1_CRM dispose d'une forte contrainte temporelle et se voit dans l'obligation de sélectionner le premier standard libre correspondant de près ou de loin à l'utilisation visée.

Dans le premier cas, les standards rescencés sont stockés dans la mémoire et le choix s'effectuera lorsqu'un standard correspondra de manière optimale à la demande de l'utilisateur ou au pire, s'en approchera le plus. Dans le deuxième cas, les conditions d'utilisation font que le choix du standard à utiliser doit être effectué dans les plus brefs délais quitte à ne pas bénéficier de la meilleure performance. On peut notamment prendre le cas des appels en urgence pour ce type de cas d'utilisation.

Résultats

Nous présenterons tout d'abord l'évolution des attributs en partant du niveau de hiérarchie le plus bas pour remonter jusqu'au L1. Les résultats seront présentés suivant le mode présenté dans le scénario 1.

L'architecture déployée pour ce scénario est présentée par la figure 4.35.

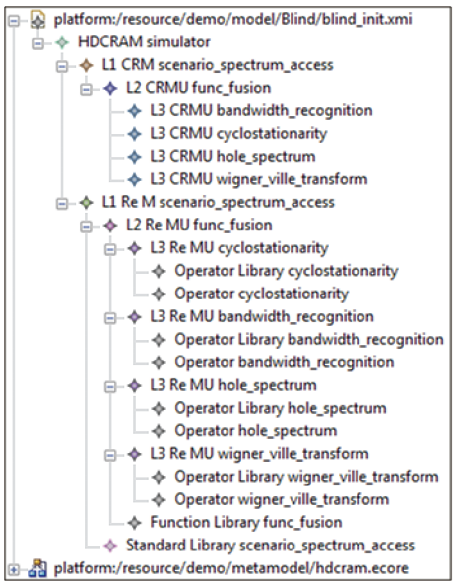


FIG. 4.35: *Ensemble d'éléments instanciés*

Pour ce scénario nous avons choisi de présenter une reconnaissance du standard bluetooth par le système. La phase de mise en œuvre du standard ne sera pas présentée, nous nous focaliserons sur la phase de recherche et interprétation des données. De plus nous ne proposons pas ici l'intégration du module de recherche ainsi que la reconfiguration de chemin de données permettant de fournir à l'ensemble des éléments de traitement du module d'analyse le signal fourni par l'élément *Bandwidth adaptation*. En effet, dans le fonctionnement, aucun signal n'est envoyé au module d'analyse tant que le module de recherche n'a pas suffisamment discriminé le signal d'entrée. De même aucun signal du module de recherche n'est envoyé au module d'analyse tant que ce dernier n'a pas fini son interprétation du signal discriminé précédent. Pour le déroulement de ce scénario nous ferons donc l'hypothèse que le signal a été précédemment discriminé, que la reconfiguration du chemin de données à été mise en place et que tous les éléments de traitement de module d'analyse sont prêts à fonctionner. De plus, nous nous placerons dans un cas d'utilisation ou le système prendra une décision de reconfiguration dès qu'il aura trouvé un standard permettant l'envoi de données (type fichier) à un équipement distant à faible distance.

Nous allons tout d'abord présenter l'ensemble des couples L3_ReMU/L3_CRMU/Operator/Library permettant le traitement du signal discrétisé par le module de recherche.

La figure 4.36 présente le bloc permettant la reconnaissance de la bande discriminée.

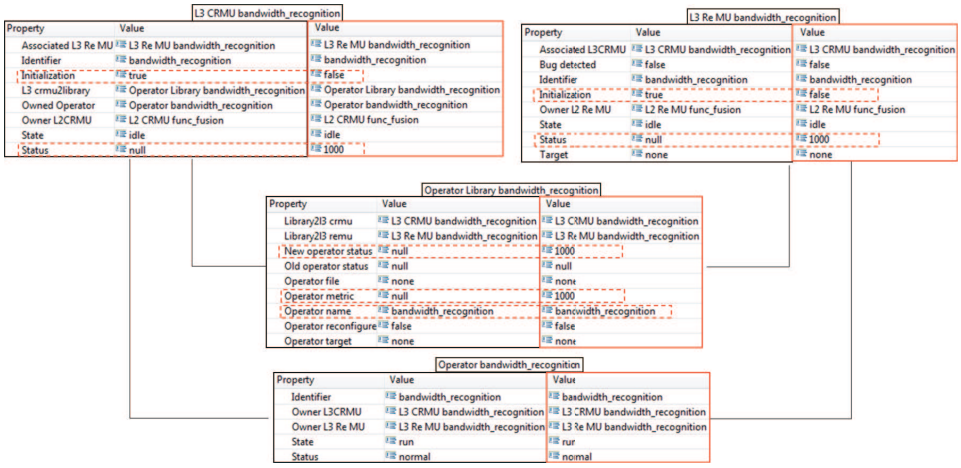


FIG. 4.36: *Vue du module L3 Bandwidth_adaptation*

Dans l'exécution du scénario, l'utilisateur rentre manuellement une fréquence en kHz, néanmoins dans un carde réel d'application, cette métrique sera directement fournie par l'opérateur. La figure 4.37 présente le bloc détectant la forme d'onde du signal discriminé.

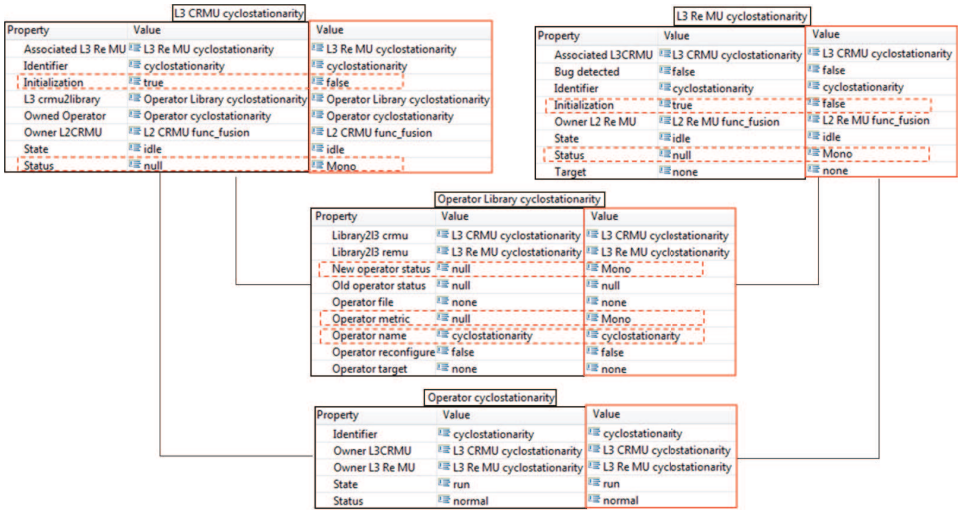


FIG. 4.37: *Vue du module L3 Cyclostationnarity*

Dans l'exécution du scénario, l'utilisateur rentre manuellement la forme d'onde : Mono pour mono porteuse et Multi pour multi porteuse. Néanmoins dans un carde réel

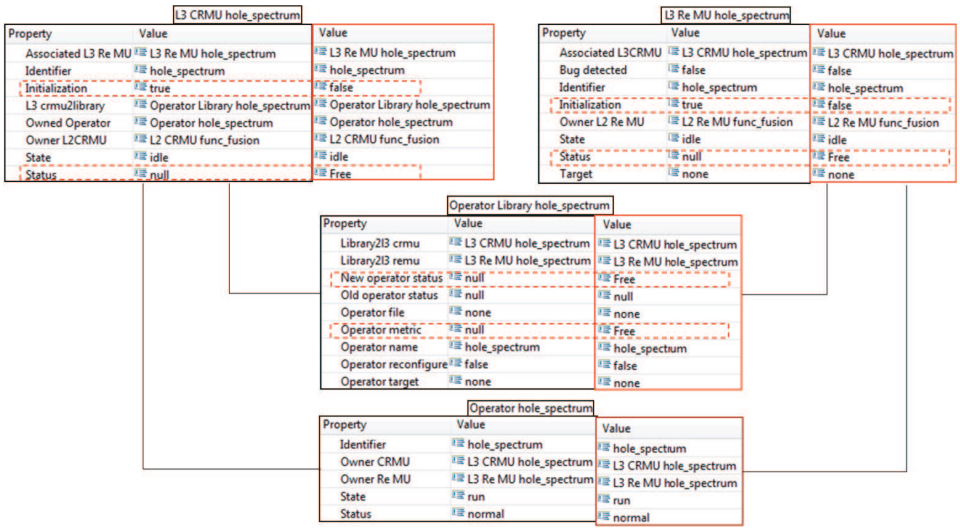


FIG. 4.38: *Vue du module L3 Hole_spectrum*

d'application, cette métrique sera directement fournie par l'opérateur. La figure 4.38 présente le bloc détectant si le signal discriminé est une bande libre ou non.

Dans l'exécution du scénario, l'utilisateur rentre manuellement l'utilisation de la bande : Free si la bande est libre, Taken si la bande est prise. Néanmoins dans un carde réel d'application, cette métrique sera directement fournie par l'opérateur. La figure 4.39 présente le bloc déterminant le mode de propagation du signal dicriminé.

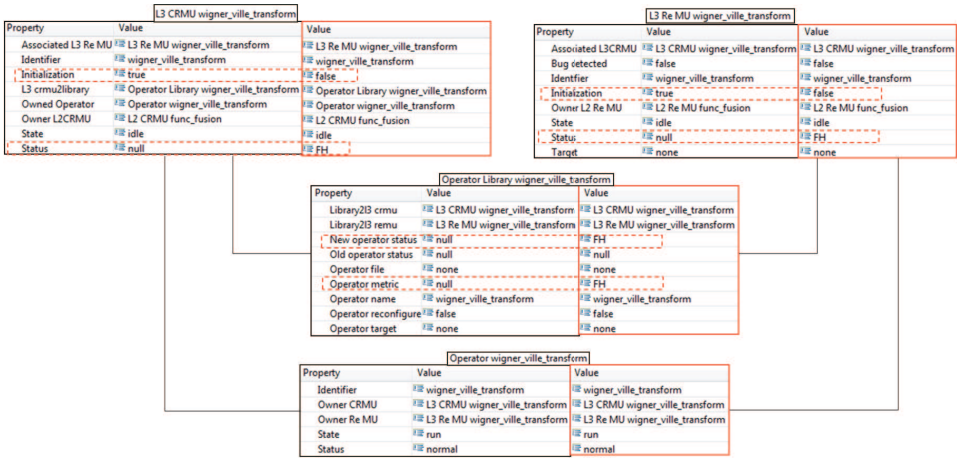


FIG. 4.39: *Vue du module L3 Wigner_ville_transform*

Dans l'exécution du scénario, l'utilisateur rentre manuellement le mode de propagation : FH pour saut de fréquence et DS pour séquence direct. Après avoir présenté l'ensemble des blocs L3 participant à l'acquisition des différentes métriques, on remarque qu'il n'y a pas eu d'interprétation des métriques acquises par les différents L3_CRMU,

les gestionnaires ont simplement effectués une vérification de la teneur de la métrique et l'ont transmise. En effet, comme la décision dépend de métriques issues de plusieurs opérateurs, les différents L3_CRMU en jeux se contentent de remonter la métrique au gestionnaire de niveau hiérarchique supérieur. Nous allons maintenant nous intéresser au gestionnaire d'intelligence de hiérarchie supérieure. Ce L2_CRMU est présenté par la figure 4.40.

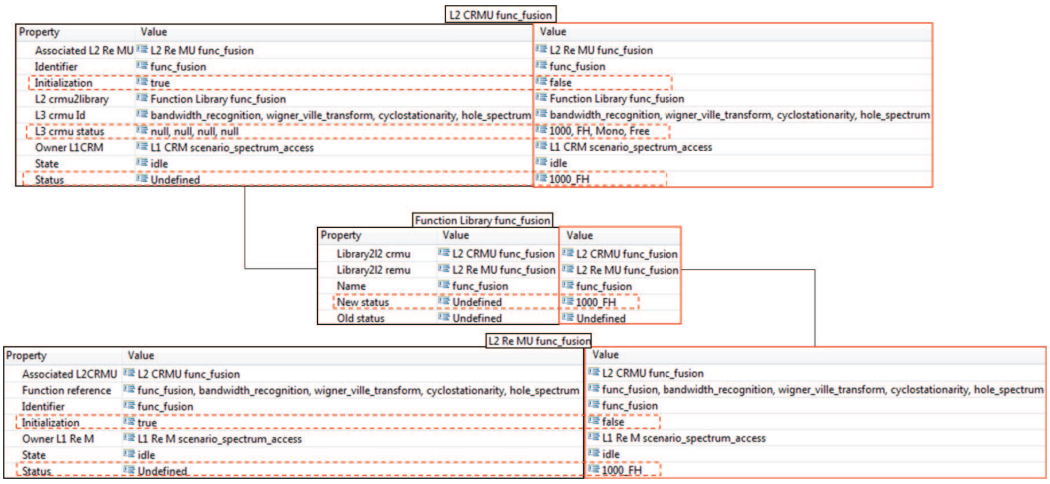


FIG. 4.40: *Vue du module L2*

L'ensemble des métriques envoyées par les différents L3_CRMU à charge sont regroupées dans l'attribut "l3_crmu_status" en concordance avec la liste "l3_crmu_Id". Ce qui veut dire que l'attribut 1 dans la liste "l3_crmu_Id" (représentant une unité L3_CRMU) a fourni la métrique occupant la première place de la liste "l3_crmu_status". Le rôle du L2_CRMU ici est de vérifier les métriques fournies et s'assurer qu'il n'y a pas d'incohérence. Si tel est le cas, une nouvelle demande de recherche sera effectuée à travers le L2_ReMU associé. À ce niveau de hiérarchie, une interprétation va être effectuée suivant l'importance relative des métriques puis transmise au L1_CRM. Ce dernier est illustré par la figure 4.41.

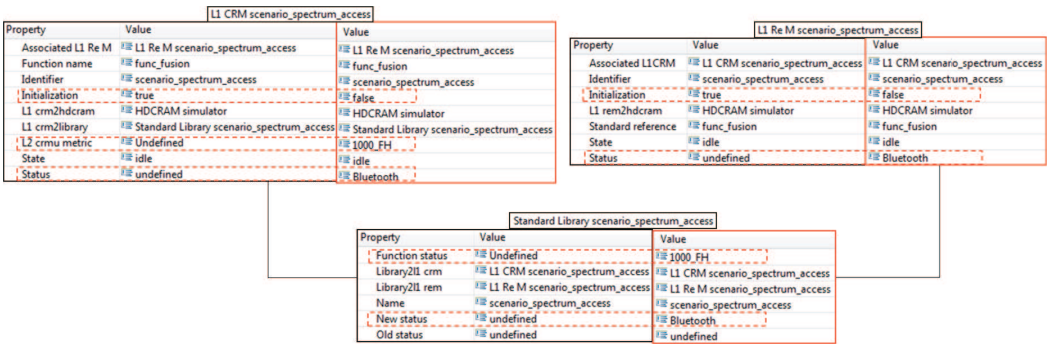


FIG. 4.41: *Vue du module L1*

Il apparaît dans l'attribut "status" de ce gestionnaire que le standard détecté suite à l'envoi de la métrique L2_CRMU func_fusion dont la valeur est accessible par l'attribut "l2_crmu_metric" est bien de type Bluetooth. La prise de décision qui découle de cette découverte est son utilisation. En effet, comme précisé dans les conditions de scénario, le but est ici de rechercher un standard permettant l'envoi de fichiers à un équipement distant. Le bluetooth remplissant ce rôle, la décision est prise de le mettre en œuvre comme illustré par la figure 4.42.

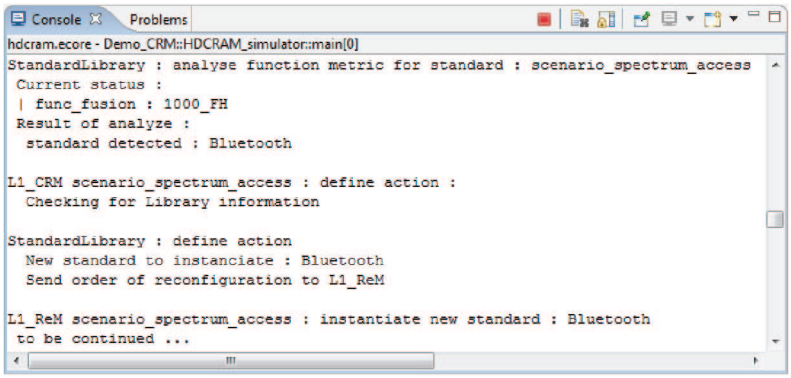


FIG. 4.42: *Sortie du simulateur - prise de décision du L1_CRM*

La phase permettant la création du standard n'est pas ici présentée et reste à concevoir dans le simulateur. Comme la phase de déploiement n'est pas critique et peut prendre plusieurs secondes, on ne voit pas d'inconvénient majeur si la puissance de calcul de l'équipement peut supporter une chaîne de transmission Bluetooth. On suppose d'ailleurs que ce genre de déploiement est parfaitement prévu et qu'il existe un scénario de déploiement préconçu. Et cela pour tous les standards que peut avoir à supporter l'équipement.

Résumé

Nous avons montré dans ce scénario de quelle façon un capteur de reconnaissance de standard en aveugle pouvait être intégré dans HDCRAM. Par l'intermédiare de ces différents getsionnaire de l'intelligence, l'information, relative au signal discriminé, est relayée du niveau le plus bas jusqu'au L1. Pendant ce trajet, cette information sera testée puis validée et permettra la reconnaissance d'un standard dans la proximité de l'équipement. Une fois le choix du standard à utiliser effectuer par l'entité L1_CRM, les informations nécessaires à son établissement seront transmises au L1_ReM afin de le mettre en œuvre et de permettre son utilisation.

Les considérations de traitement du signal associées à ce scénario font l'objet de la thèse de Rachid Hachemani, autre doctorant de l'équipe. Mon travail vise à définir l'architecture de gestion à y associer pour rendre ce scénario viable dans un équipement RC.

Afin de permettre à un équipement d'intégrer les fonctionnalités introduites par ce scénario, il lui faut disposer des éléments présentés dans les deux tableaux 4.6, 4.7 ainsi que 4.8 suivants.

TAB. 4.6: *récapitulatif des éléments CRMU intervenant dans l'exécution du scénario*

Unité	Opération	Descriptif
L3_CRMU • bandwidth recognition • wigner ville transform • cyclostationnarity • hole spectrum	capture_metric	Permet l'acquisition de la métrique en provenance de l'opérateur associé
	analyze_metric	Pas d'interprétation, test sur la validité de la métrique
	define_action	Définit l'action à réaliser • Si ErrorMetric alors faire nouvelle acquisition • Sinon transmettre à L2_CRMU func_fusion
L2_CRMU func_fusion	capture_metric	Permet l'acquisition de la métrique en provenance de chaque L3_CRMU associé
	analyze_metric	Vérification des métriques puis interprétation
	define_action	Définit l'action à réaliser • Si ErrorMetric alors faire nouvelle acquisition • Sinon transmettre à L1_CRM
L1_CRM	capture_metric	Permet l'acquisition de la métrique en provenance L2_CRMU func_fusion
	analyze_metric	Interprète la métrique et en déduit le type de standard ainsi que sa possible utilisation
	define_action	Prend la décision d'utiliser le standard ou attend de nouvelles métriques afin d'optimiser le choix du standard aux besoins utilisateur
	send_order2L1_ReM (order : abstract_order)	Transmet l'ordre de reconfiguration au L1_ReM associé qui prendra en charge l'implantation du nouveau standard

TAB. 4.7: *récapitulatif des opérations du L1_ReM intervenant dans l'exécution du scénario*

Unité	Opération	Descriptif
L1_ReM	receiveL1_CRMorder(order : abstract_order)	Permet l'acquisition de l'ordre abstrait de reconfiguration du L1_ReM associé
	interpreteOrder(order : abstract_order)	Interprète l'ordre de reconfiguration : Si changement de standard alors suivant ce qui est déjà implanté, décide des reconfigurations • reconfigureL2_ReMU • create L2_ReMU • finalize L2_ReMU
	reconfigureL2_ReMU (target : crm_reference, order : reconfiguration_order)	Cible le L2_ReMU à reconfigurer et lui transmet un ordre abstrait de reconfiguration
	create_L2_ReMU (NewIdentifier : rem_reference, NewStatus : status)	Création d'un nouveau L2_ReMU pour la mise en œuvre du nouveau standard de communication
	finalyzeL2_ReMU(target : L2_ReMU)	Cible et drétruit le L2_ReMU n'étant plus nécessaire (attend la validation par le L2_ReMU de la possibiilté de destruction)
	reconfigureDataPath()	Reconfiguration du chemin de donnée si nécessaire

TAB. 4.8: *récapitulatif des opérations des L2_ReMU et L3_ReMU intervenant dans l'exécution du scénario*

Unité	Opération	Descriptif
L2_ReMU func_fusion	receiveL1_ReMorder(order : abstract_order)	Permet l'acquisition de l'ordre abstrait de reconfiguration du L2_ReMU associé
	interpreteOrder(order : abstract_order)	Interprète l'ordre de reconfiguration : • Si reallocate : met en place le changement de cible (création d'un nouveau module Operator_1 et modification du chemin de données)
	reconfigureL3_ReMU(target : crm_reference, order : reconfiguration_order)	Cible le L3_ReMU à reconfigurer et lui transmet un ordre abstrait de reconfiguration
	create_L3_ReMU (NewIdentifier : rem_reference, Newtarget : OperatorTarget, NewStatus : status)	Création d'un nouveau L3_ReMU
	finalyzeL3_ReMU(target : L3_ReMU)	Cible et drétruit le L3_ReMU n'étant plus nécessaire (attend la validation par le L3_ReMU de la possibilité de destruction)
	reconfigureDataPath()	Reconfiguration du chemin de donnée si nécessaire
L3_ReMU	reconfigure	• Reçoit l'ordre de reconfiguration • Ordonnance l'arrêt de l'opérateur à charge
	data_base_access	Interprète l'ordre de reconfiguration : • Chargement d'un fichier de configuration
	finalyze()	• Détruit son opérateur associé ainsi que son L3_CRMU associé.

4.6 Synthèse des scénarios

Nous venons de parcourir 3 scénarios qui présentent les cas de fonctionnement qu'un équipement RC sera amené à rencontrer en cours d'utilisation. Ces principaux cas sont :

- la reconfiguration du chemin de données entre opérateurs de traitement qui peut être utile afin de :
  - court-circuiter un opérateur non essentiel au traitement,
  - réallouer un opérateur de traitement sur une nouvelle cible matérielle,
  - participer à la mise en place d'un hand over vertical sans interruption de service.
- La prise en compte du niveau de batterie qui peut avoir comme effet la modification d'un certain nombre d'éléments de traitement.
- La reconnaissance de standards à proximité de l'équipement.

Parmi les scénarios présentés, le scénario 1 peut rappeler un algorithme qui n'est pas nouveau en soit puisqu'il s'agit du fonctionnement d'un algorithme adaptatif. Néanmoins, utilisé de manière classique, les décisions de modifications effectuées par un tel algorithme ne sont pas mises à disposition de l'ensemble de l'équipment. C'est justement ce que permet HDCRAM. Un changement est automatiquement transmis et pris en compte dans les niveaux de hiérarchie supérieure ce qui peut permettre de croiser l'adaptation effectuée par l'algorithme à une ou plusieurs autres reconfigurations afin d'améliorer davantage l'adaptation.

Nous sommes conscients que la façon de traiter les scénarios est actuellement simpliste. Il est clair que le scénario de gestion du niveau de la batterie nécessite un ensemble de stratégie globales et que les leviers d'actions ne se résument pas à un seul opérateur. Il en est de même pour le nombre de métriques à prendre en considération amenant à une prise de décision de reconfiguration. On peut pour cela imaginer, dans un premier niveau de complexité, l'utilisation de règles pré-établies. Ainsi un gestionnaire (L2_CRMU ou L1_CRM) disposerait d'une table présentant en entrée les métriques relevées. Suivant le jeux de paramètres, ce gestionnaire fournirait en sortie une reconfiguration adaptée à la situation. Avec une complexité plus élevée, c'est à ce niveau que l'on pourrait bénéfier de la capacité d'auto apprentissage qui permettrait de noter les reconfigurations sélectionnées suivant leur performance. À partir de ces notes le gestionnaire pourrait alors sélectionner au mieux le type de reconfiguration à effectuer suivant les cas préalablement rencontrés. Cette approche fera l'objet d'une autre thèse qui démarre sur ce sujet.

Nous n'avons pas présenté de scénario illustrant la communication de HDCRAM avec l'extérieur. Néanmoins nous rappelons que dans HDCRAM, un des rôles du L1_CRM est justement d'offrir une interface avec l'extérieure permettant ainsi de relever les demandes quelles soient de l'équipement vers le réseau ou du réseau vers l'équipement. Nous rappelons ainsi que notre architecture permet de supporter une approche terminal centric ou network centric. Et surtout si, comme si nous le pressentons, un mélange des

deux approches est plutôt utilisé, HDCRAM offre tous les services permettant de les supporter toutes deux.

4.7 Conclusion

NOUS avons présenté dans ce chapitre un ensemble de trois scénarios pouvant être rencontrés par un équipement RC. Nous y avons introduit l'environnement de simulation ainsi que l'utilisation du simulateur pour l'exploration d'architectures à déployer dans un équipement RC. Aux vues des différents tableaux présentés en résumé à la fin de chaque scénarios, nous avons pu remarquer que les fonctions à utiliser pour le bon déroulement des scénario sont identiques d'un scénario à l'autre. Seul le comportement de ces fonctions change suivant les cas à traiter. Nous pouvons donc affirmer avoir défini au niveau fonctionnel l'ensemble des opérations pour chaque élément de l'architecture hiérarchique et distribuée permettant à un équipement RC de s'adapter aux variations de l'environnement. Ceci tant en termes de gestion d'éléments n'offrant pas de possibilité de reconfiguration, mais participant à la remontée d'information aidant à la prise de décision, que de gestion d'éléments reconfigurable offrant ou non des services RC.

En effet, nous avons également joué d'autres scénario qui ont mis en évidence que chaque élément CRM(U) de HDCRAM doit posséder les opérations suivantes :

- capture_metric,
- analyse_metric,
- define_action,
- send_order2associateReM,
- send_metric2ownerCRM.

De même chaque élément ReM(U) doit posséder les opérations suivantes :

- receive_order,
- interprete_order,
- reconfigure,
- create,
- finalize.

L'ensemble de ces opérations utilisées dans les différents éléments constituant HDCRAM permet à un équipement RC de répondre aux attentes en termes d'adaptation par une prise de décision rapide et efficace ainsi qu'une gestion fine des possibilités de reconfiguration de l'équipement. C'était le but fixé il y a trois ans au démarrage de cette thèse et nous pouvons le considérer comme atteint.

Conclusion générale

NOUS avons présenté dans ce mémoire une architecture adaptée au domaine de la radio cognitive reposant sur une approche interdisciplinaire mettant en jeu les domaines de l'électronique, du traitement du signal, des sciences cognitives ainsi que de l'informatique. Une telle architecture se doit d'offrir une reconfigurabilité logicielle étendue permettant l'adaptation constante de l'équipement vis-à-vis des changements permanents de son environnement et des besoins utilisateurs. Cette adaptation est possible grâce à une gestion d'un ensemble de capteurs qui permet à l'équipement de prendre conscience de son environnement. Les signaux ainsi acquis sont traités par différents algorithmes plus ou moins complexes, qui reproduisent le fonctionnement du cerveau en réagissant à l'ensemble des informations qu'il reçoit par une prise de décision permettant ainsi l'adaptation du mode de fonctionnement à son environnement. On peut ainsi prendre en exemple l'utilisation de réseaux de neurones offrant à l'équipement des capacités d'apprentissage vis-à-vis de ces précédentes décisions. Afin d'offrir une reconfigurabilité étendue à un équipement radio, il faut mettre en place une gestion fine de la plateforme d'exécution et donc gérer efficacement l'hétérogénéité des ressources ainsi que leurs disponibilités. Cette hétérogénéité des ressources est efficacement gérée par notre architecture qui présente une hiérarchie distribuée sur trois niveaux d'abstraction offrant de plus une possibilité de reconfigurations à différents niveaux de granularités. La modélisation de notre architecture en langage UML permet une ouverture à la communauté scientifique par son langage universel et une indépendance vis-à-vis des langages d'implantation et des contraintes purement techniques. L'utilisation d'un métalangage de programmation exécutable dans notre métamodèle nous permet de clairement spécifier la sémantique opérationnelle de notre architecture par la façon dont sont effectuées les captures de métriques, leurs interprétations, les prises de décision puis les reconfigurations. Par la création d'interfaces de communications communes, il est ainsi aisé de réutiliser des blocs de traitement existant en les intégrant à notre architecture. De fait, le simulateur est une aide à la conception d'équipement RC et peut être intégré dans le flot de conception.

De plus, la distribution hiérarchique des gestionnaires permet une adaptation efficace aux évolution de l'environnement ainsi qu'une prise en compte des particularités de chaque composant de l'équipement. En effet, les gestionnaires de niveau 3 étant au plus près des composants matériel ils peuvent être conçus de façon à en exploiter les moindre particularités. Les gestionnaires de niveau 2 permettent de prendre en compte l'aspect hétérogène des ressources de la plateforme d'exécution cible et ont pour rôle à la fois de consitituer une couche d'abstraction entre le niveau 1 et le niveau 3 ainsi que de

contrôler les gestionnaires de niveau inférieur. Le couple L1_CRM/L1_ReM à pour rôle la gestion globale de l'équipement ainsi que de faire l'interface avec le monde extérieur (utilisateur, réseau et entités de régulation).

L'un des points fort de HDCRAM est la possibilité d'y intégrer à la fois :

- des composants standards de traitement du signal (n'offrant ni capacité de reconfiguration ni métrique participant à la prise de décision),
- des composants reconfigurables apportant de la flexibilité (mais ne participant pas à la remontée de métrique),
- des composants présentant des capacités de reconfiguration et proposant des métriques qui participent à la prise de décision.

Il apparaît aussi que la mise en place d'une intelligence au sein d'un équipement radio (de type embarqué ou non) pour la modélisation de son environnement et sa compréhension est très complexe. Pour un équipement de type embarqué par exemple, la gestion des ressources énergétiques est un point critique. Il faut donc s'assurer que le système déployé n'est pas trop consommateur d'énergie. Or, les algorithmes de traitement tels que les réseaux de neurones, heuristiques diverses (algorithmes génétiques, recuit simulé, etc.), permettant de simuler l'intelligence, sont très exigeants en puissance de calcul et donc en terme de consommation d'énergie. On peut néanmoins envisager qu'une gestion fine des ressources par l'implantation et l'activation des blocs de traitement seulement nécessaires à la tâche demandée, à un instant donné, limitera la consommation. On peut pour cela prendre en exemple l'activation ou non des services tels que le Wifi ou la 3G qui sont, sur les équipements actuels, activés en permanence. On peut aussi prévoir l'évolution des batteries en termes de longévité comme l'utilisation de piles à combustible dans les équipements mobile comme présentée dans [110]. Ce problème est moins important sur des architectures de type station de base.

La faculté d'adaptation laissée à un équipement radio soulève une question : "une reconfiguration d'un terminal radio de type embarqué (afin d'assurer un meilleur service à son utilisateur) n'est elle pas mauvaise pour l'ensemble du réseau ?" L'adage ne dit il pas que le mieux est l'ennemi du bien ? On voit apparaître ici une notion d'objectif de performance par la prise en compte du compromis entre performance du réseau et performance du terminal. Bien que ce champ de recherche n'ait pas été traité dans cette thèse, c'est une optimisation globale, au niveau système, qui doit être faite. HDCRAM permet tout à fait de supporter cela en recevant des ordres de l'extérieur de l'équipement donc issus du réseau, ou en donnant la possibilité d'envoyer vers le réseau des informations de l'environnement de l'équipement.

La présentation de scénarios réalistes dans un contexte RC a permis de définir un ensemble d'opérations nécessaires au bon fonctionnement, et qu'il faut donc intégrer à la conception d'équipements intelligents. Bien que nécessitant la mise en place d'un certain nombre de composants logiciels ou matériels supplémentaires aux chaînes de traitement

existant, cela se révèle être le prix à payer pour offrir une flexibilité permettant l'adaptation de l'équipement de façon autonome. En effet, il faut garder à l'esprit que dans un cadre applicatif, les composants logiciels décrits dans HDCRAM (plus particulièrement les gestionnaires de niveau 3) pourront être définis par des langages de programmation de bas niveau. Ainsi, la notion d'appel de fonction est à prendre au sens large et peut tout autant être défini par le passage d'un signal dans une machine d'état ou un ensemble de portes logiques. Néanmoins la notion d'appel de fonction prend tout son sens pour une implantation des gestionnaires sur des processeurs généralistes et se rapproche ainsi des comportements logiciels tels que nous les connaissons dans le monde informatique.

La conduite de ces travaux durant mes trois années de thèse m'a permis d'appréhender les notions relatives à la radio cognitive avec un vaste champ de possibilités de recherche tant du point de vue du traitement du signal radio, des protocoles de communications, de la normalisation ou encore de la gestion de plateformes hétérogènes. J'ai eu plaisir à effectuer des recherches dans le domaine des moteurs d'intelligence même si cela ne constituait pas le cœur de mes recherches, et notamment ceux inspirés de la biologie qui tentent de dupliquer les possibilités de réflexion, d'estimation, de prise de décision ou encore d'apprentissage de l'esprit humain pour les embarquer dans un système numérique. De plus, l'utilisation d'UML pour la modélisation de HDCRAM m'a permis de prendre conscience de l'existence de l'ingénierie dirigée par les modèles (IDM) et de tout un champ du domaine de la conception. Cette méthode, dont je ne connaissais rien, pourrait permettre d'affronter les défis que peuvent représenter la complexité et l'évolution croissante des applications et du matériel radio. L'IDM permet en effet de passer d'un mode descriptif, dans lequel les modèles ne servent que de documentation, à une mode productif où ces mêmes modèles sont cette fois utilisés pour l'analyse jusqu'à la génération de code exécutable.

Perspectives

Le métamodèle que nous avons présenté permet une simulation fonctionnelle de haut niveau d'un système adapté à la radio cognitive. La future évolution est de proposer des règles de transformation de modèle de manière à prendre en compte les contraintes physiques des plateformes d'exécution. Ces transformations permettraient ainsi de simuler les concepts techniques tels que le protocole de communication, l'utilisation des ressources (répartition matériel des opérateurs, occupation mémoire), la charge processeur ou les caractéristiques matérielles inhérentes aux différentes cibles d'exécution (temps de propagation, débit maximum, etc.). Ce travail fera l'objet d'une nouvelle thèse par Istas Pratomo.

Une perspective directe de ma thèse va être exploitée par Wassim JOUINI qui vient de rejoindre l'équipe SCEE en tant que doctorant. Son travail de recherche va en partie consister en la mise en place d'algorithmes d'intelligence pour permettre la prise

de décision d'un CRM(U) suivant le contexte. Une piste d'étude serait de permettre au concepteur de remplir suivant ses propres besoins la fonction représentant l'intelligence du CRM(U) visé. Pour cela, on peut imaginer une fenêtre laissant la possibilité au concepteur d'utiliser un langage de programmation (à partir d'une liste de langages prédéfini) pour coder le moteur d'intelligence d'un CRM(U). Puis un parseur viendrait interpréter ce code en langage Kermeta. Cette interprétation permettrait de simuler au niveau fonctionnel l'ensemble du système dans des conditions précises. On peut à ce titre penser aussi à la création d'un métamodèle permettant la saisie de scénario afin d'étendre les possibilités de notre simulateur.

La phase ultime de validation de ces travaux de thèse est de permettre l'intégration de HDCRAM au sein d'une plateforme de prototypage et de valider son fonctionnement en temps réel. Il serait aussi intéressant de permettre lors de la conception une répartition matérielle automatique des opérateurs sur la plateforme cible à l'aide d'algorithmes mathématiques. Cette répartition automatique pourrait être intégrée avec le métamodèle dans un outil de conception ayant des liens avec des outils existant permettant des simulations avancées du système créé. Il deviendrait ainsi possible de gérer la conception d'un équipement radio en apportant l'aspect description à haut niveau pour s'abstraire des contraintes techniques du matériel jusqu'à la validation matériel sur la plateforme d'exécution cible par une génération automatique de code. Un tel outil est très intéressant car il permet d'intégrer simplement les nouvelles technologies disponibles (qu'elles soient logicielles ou matérielles) et d'assurer une rétro compatibilité avec le parc d'équipement existant.

Dans une perspective liée aux différents travaux effectués dans l'équipe SCEE il serait aussi envisageable de permettre la modélisation des acquis dont l'équipe dispose sur les différentes thématiques de recherche. On peut par exemple nommer la reconfiguration partielle dynamique, le PAPR, la détection de bandes libre ou encore l etraitement d'images afin de les capitaliser. De même il serait aussi possible de permettre une modélisation des différents travaux menés sur les plateformes de prototypage afin de s'abstraire des contraintes d'un unique fournisseur. Ceci offrirait la liberté de changer de plateforme sans voir apparaître les problèmes de mise à niveau de tous les travaux effectués et indubitablement éviter la perte des plus anciens. Il faut tout de même noter certaines incompatibilités inhérentes au matériel ainsi une reconfiguration partielle par différence ne pourra (dans l'état actuel de la technologie) pas s'effectuer sur une autre cible matérielle qu'un FPGA de la famille Xilinx (à partir du Virtex II Pro). Néanmoins l'évolution vers une plateforme offrant les nouvelles générations de composants pourra s'obtenir de manière automatique. Un bénéfice collatéral est que ce genre de modélisation nécessite la mise en place d'interfaces communes. Une fois celles-ci définies, il sera possible de regrouper l'ensemble des IP conçues dans l'équipe au sein d'un même système ce qui permettra de traiter des cas de reconfiguration plus complexe.

Annexes

Annexe 1

NOUS nous proposons ici de présenter l'outil de modélisation Kermeta (KERnel META-modelling). L'équipe responsable du développement de Kermeta se nomme la Triskell team et officie à l'IRSIA[1] de rennes sous la direction de Jean-Marc Jézéquel, professeur à l'Université de Rennes 1. L'objectif de cette équipe est de fournir aux concepteurs logiciels un ensemble d'outils et de bibliothèques de composants spécifiques afin de permettre la spécification de propriétés à la fois fonctionnelles et non fonctionnelles.

L'objectif de Kermeta est d'offrir un dénominateur commun à l'ensemble des techniques relatives à l'utilisation de modèles. Il doit donc proposer un ensemble de technique comme :

- définir des métamodèles,
- instancier ces métamodèles,
- spécifier leur sémantique opérationnelle,
- exprimer des contraintes,
- etc.

Le but de ce dénominateur commun n'est pas de remplacer l'ensemble des langages dédiés utilisés mais de fournir une base sémantique commune à ces langages afin de leur permettre d'interopérer naturellement. En effet on peut nommer pour les plus utilisé : EMOF pour les métamodèles, OCL pour les contraintes ou encore Java pour la définition de la dynamique des opérations. Dans un tel contexte où cohabitent plusieurs langages hétérogènes pour la définition d'un système, on augmente fortement les risques techniques liés au manque d'interopératbilité des langages utilisés ce qui peut amener à des problèmes de vérifications entre les différentes réalisations de chaque langages. Prenons l'exemple d'un métamodèle dont les contraintes sont en OCL et la sémantique exprimée en langage naturel. Il devient extrêmement difficile de vérifier la cohérence du résultat d'une transformation de modèle exécutée en Java (par exemple).

D'où l'intérêt principal de l'approche Kermeta qui est de permettre la définition les différents aspects relatifs à un métamodèle de façon homogène. Un métamodèle peut par

[1] http ://www.irisa.fr/home_html

exemple encapsuler directement les contraintes qui lui sont attachées et la spécification de sa sémantique opérationnelle. Le noyau de métamodélisation proposé se présente sous la forme d'un métalangage orienté objets construit comme une extension de EMOF. La figure 4.43 représente le concept de Kermeta.

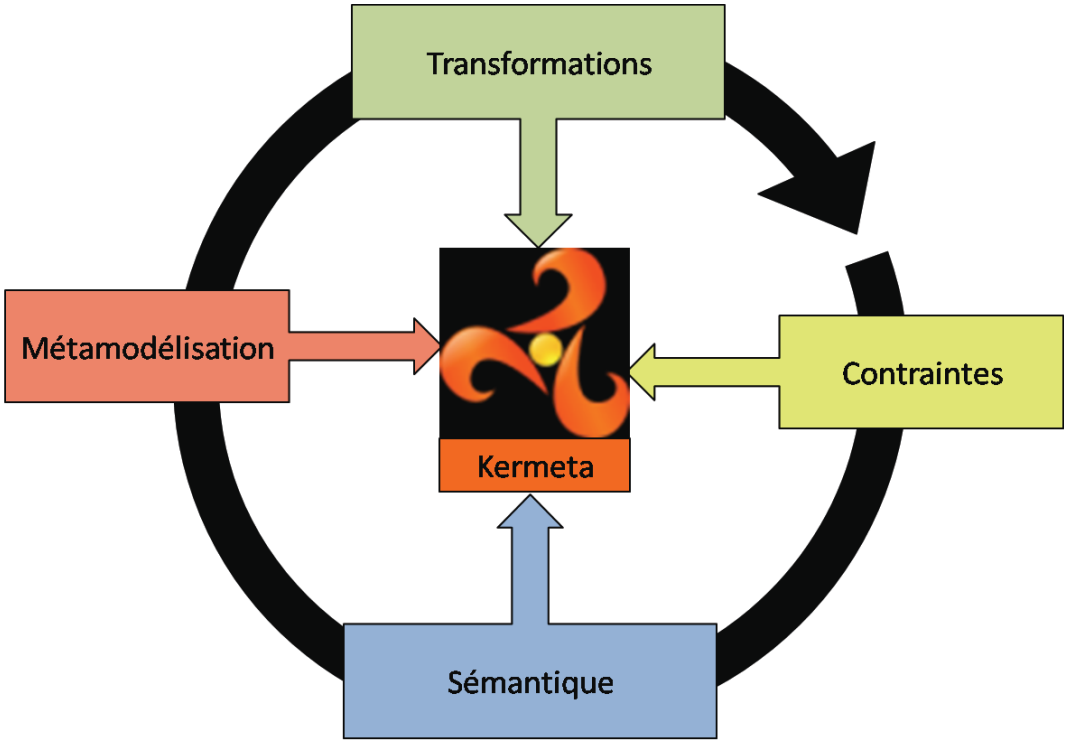


FIG. 4.43: *Kermeta, un cœur commun*

Pour plus de précision sur Kermeta voir la thèse de Franck Fleurey [111] actuellement post doctorant au sein de l'équipe Triskell ou encore l'article très complet [112].

Annexe 2

NOUS nous proposons ici de présenter le rapport émis suite à la conclusion du réseau d'excellence Newcom pour le groupe de travail DRD7 du projet D sur la radio flexible. Ce délivrable présente les objectifs du projet ainsi que les moyens mis en œuvre pour les accomplir.

NEWCOM

Network of Excellence in Wireless COMmunications

Contract No IST NoE 507325

DRD.7 – Report on HW/SW architecture of the Flexible Digital Platform

Contractual Date of Delivery to the CEC:	28/02/2007
Actual Date of Delivery to the CEC:	5/03/2007
Author(s):	UPC: Antoni Gelonch, Ismael Gomez SUPELEC: Christophe Moy, Loig Godard TUA: Martin Witte
Participant(s):	SUPELEC, UPC, TUA
Workpackage:	*WPRD Research integration for Project D "Flexible Radio""*
Est. person months:	
Security level:	CO
Nature:	Report
Version:	2.0
Total number of pages:	29

Abstract:

In this deliverable is described a common project developed to integrate the different methodologies assumed for the different partners to provide flexibility to the digital HW/SW platforms for wireless systems. The initial approach, the main elements and details of the implementation done and the solutions proposed can be found in the document.

Keyword list: Flexible radio, reconfigurable radio, software defined radio, cognitive radio

DOCUMENT REVISION HISTORY

Version	Date	Author	Summary of main changes
01.0	15-11-2006	Antoni Gelonch, Ismael Gomez	First draft with inputs from UPC
01.1	29-11-2006	Christophe MOY	SUPELEC first addings
01.2	12-12-2006	Loig Godard	SUPELEC contribution
01.3	29-01-2007	Antoni Gelonch, Ismael Gomez	UPC contribution
02.0	15-03-2007	Antoni Gelonch	Final version

CONTENT

1 INTRODUCTION

The market for digital communication systems is constantly expanding with the development of new services and applications. Trying to provide the necessary capabilities, the future 4G wireless systems are envisaged as a set of different radio access technologies with overlapped coverage areas offering different, but defined, multimedia service classes. In addition the mobility pattern has experienced important changes that suppose new management paradigms. This wide variety of conditions has led to an explosion of communication standards with different modulation schemes, channel bandwidth, dynamic ranges, etc. In the other side, users are demanding for low cost, low power, and small size systems to satisfy these new communication standards in most of the cases considering a multistandard platform. All this aspects strongly demand high levels of flexibility on the communication system side to adapt to the variable operational conditions and a high percentage of these flexibility requirements are translated to the equipment that supports such systems. Therefore, one of the most challenging research areas is related with the development of flexible terminals capable to roam seamlessly among different wireless systems where the Software Defined Radio concept is one of the most visible approaches.

1.1 FLEXIBILITY CONCEPT

1.1.1 DEFINITION OF FLEXIBILITY

The definition of the flexibility concept is not an easy task. It must introduce in the discussion all the aspects, also those that do not have a clear relation among them, that provide the capacity to modify the system to adapt its behaviour to the variable conditions of the environment. In that definition we introduced to verbs, modify and adapt, that express clearly what is expected from a flexible system. Is time now to talk about methodology and therefore we should introduce the concept of modularity. Developing the system following a modular approach in software but also in hardware introduces the first step to achieve flexibility. A modular approach as a methodology of hardware design allows changing a board for another with different or improved characteristics for the current situation without substituting the complete system. A modular approach as a methodology in software design allows the easy substitution of the software pieces that compound the complete system. Then, substitution can be considered as the complete solution to provide flexibility although in most of the cases only few changes are necessaries to adapt the system to the environment conditions. But in a complex system there are other inputs that we must consider when talking about flexibility and all them are related with the cost that requires to obtain a certain level of flexibility. We can notice that we introduced two additional ideas: the flexibility levels and the cost to achieve flexibility. All this aspects will be treated in the following sections.

1.1.2 FLEXIBILITY LEVELS.

It has been stated before that flexibility is only available at increasing the system cost due to increasing the energy consumption, the chip area or the execution speed. Therefore is mandatory identifying the flexibility levels, basically understood as reconfiguration levels, and the costs to achieve each one of them. Three basic levels of reconfiguration are envisaged:

- Parametric Reconfiguration: Implies the dynamic alteration of individual parameters of signal processing functionalities (e.g. changing some parameter of an specific module).
- Structural Reconfiguration: Considers the alteration of the layout of the radio system substituting one modules for another (e.g. change modulation scheme).
- Application Reconfiguration: Assumes a completely replacement of the radio application radio (e.g. replacing GSM for WLAN).

Such levels or flexibility are more focused with the management of the system. In fact we can consider them flexibility in management phase. Under such concept the relevant issues relay in the reconfiguration framework that should be able to provide a complete access to the hardware and full capacity to reconfigure them in run-time.

In addition, we must also talk about flexibility when dealing with the system design phase. Flexibility in the design phase must take into account the aspects related with the code reuse, common language for describing a system, common simulation, development and optimization tools, common framework where the code will be executed, etc. all them of high relevance for designing efficient HW/SW systems suitable for wireless environment.

2 COMMON APROACH

A common design of a SoC is proposed to define a common methodology merging the TUA, SUPELEC and UPC approaches. This design has been defined to integrate the different methodologies from each partner. Therefore he has been based in the SoC design methodology from TUA incorporated in a commercial design tool, the LISATECK tool, the use of the Platform-Hardware Abstraction Layer (P-HAL) concept from UPC and in the definition of common operators and the incorporation of hierarchical configuration management structures developed by SUPELEC. The goal is to merge, by means of a simple implementation, the different point of view of the partners and to show the benefit of the of reconfiguration concept at design level and also at run-time management.

2.1 PLANNED COMMON PROJECT DESCRIPTION

2.1.1 FLEXIBILTY APPROACH

2.1.1.1 Flexibility at design time: LISATECK tools

The flexibility required by *Software Defined Radio* or *Reconfigurable Radio* is only available at the costs of power consumption, chip area and execution speed. One extreme of this trade-off is the field of General Purpose Processors which provide high flexibility but only with a very high power consumption and low performance. The other extreme is the domain of Application Specific Integrated Circuits (ASICs) with low power consumption and high performance but almost no flexibility. As shown by current research trends **Erreur ! Source du renvoi introuvable.**, reconfigurable application specific programmable architectures are promising candidates for solutions that fulfill power and performance as well as flexibility requirements.

The use of tools like Language for Instruction Set Architectures (LISA) can reduce the design process and introduce high flexibility in the design phase. LISA is an Architecture Description Language (ADL) which allows – on a high abstraction level - the joint description of behaviour and timing of programmable architectures, their simulation and validation, automatic generation of synthesizable Register Transfer Level (RTL) descriptions as well as automatic generation of software development tools like compiler, assembler, linker and simulator. Thus, LISA is a powerful framework for the rapid development of dedicated programmable architectures with complex application specific instructions including a complete design space exploration.

During the design space exploration the application is analysed, specialized instructions are identified, implemented and refined depending on the results of instruction set simulations. Thus ASIPs open the door for programmable architectures with a high performance and low power consumption required by *Software Defined Radio*. But the flexibility needed by changing communication standards requires behavioral changes that can not be achieved efficiently by only rewriting an application on an ASIP. Therefore reconfigurable hardware elements need to be integrated into the ASIP structures in order to reconfigure the ASIP in a way that it is suited for each individual application. However, common functionality that is required by all algorithms, e.g. instruction fetch mechanism or control instructions can still be implemented as dedicated hardware to guarantee the optimum efficiency.

2.1.1.2 Providing flexibility in run-time: P-HAL

To develop SWR applications able to run on different heterogeneous hardware/software platforms not worrying about the actual configuration, some kind of abstraction layer should be described. In addition it is assumed that the radio application is then built from a set of independent blocks (each one may be a whole algorithm or just a part) that exchange information through virtual interfaces. Blocks are programmed in a non contextualised way so that they can be easily inserted or removed from the system.

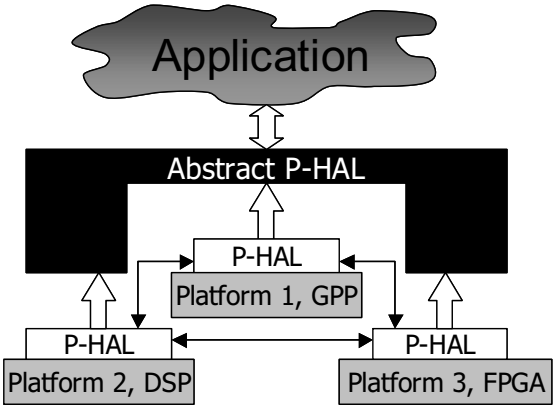


Figure 2-1. Abstraction Layer

The main functions associated with the use of such kind of abstraction layers includes the capability to perform:

1) Seamless communication. Transparent packet exchange mechanisms between on-board and off-board application blocks.
2) Monitoring and control. Each block may show the value of internal parameters (and even allow its modification) or other relevant information.
3) Real-time. Timing control to of any application objects to get the applications properly working. This includes synchronisation of individual platforms and CPU assignment to different objects in a time-sharing environment.
4) Object launch and map. Each one of the application objects has to be assigned one of the real processors (mapping), downloading the executable code on such processor and then start its execution.

Therefore the flexibility provided by such layer [4] includes the capability to manage the radio application, block by block, initiating and stopping its execution and performing computing resources management of the complete platform by proper mapping of each block in one of the processors.

2.1.1.3 Flexibility in Management: Common operators and management entities

For a maximum of efficiency, the management of the flexibility has to be spread over the system. Several levels of management must be involved, depending on the scale of the reconfiguration operations in the system. This also permits to consider reconfiguration at several levels of abstraction. From a high level (called L1), decorrelated from the HW implementation for genericity purposes, down to a low level (called L3), very connected to the HW implementation for implementation efficiency purposes. Some hierarchy relationship links these two levels through an intermediate medium level (called L2) of management. That is SUPELEC approach for the reconfiguration management of flexible radio systems [1]. It lies on a distributed and hierarchical management on 3 levels. It has been defined to adapt to heterogeneous HW structures usually met in flexible radio systems. It is indeed particularly challenging to cope with the reconfiguration of a sub-part of a system that is distributed on a variety of devices such as DSP, GPP, FPGA, and ASIC [2]. The extension of this approach towards the design of future cognitive radio systems is already started and promises very interesting advances [3].

2.1.2 COMMON DESIGN SoC CONCEPT

A very basic design of a turbo-coder is envisaged. We can not afford designing a too much complicated system in a so short period with so many kinds of technologies or design methods to involve.

The principle is the following. Use each partner specific approach of reconfigurability in the same design. In other words, implement in a SoC made of ASIPs, several (3) common operators with parameterization features on a P-HAL abstraction supporting a hierarchical reconfiguration management. We will draw some conclusion hereafter to generalize the approach.

One of the considered scenarios is the design of a turbo-coder. This can be achieved using only one coder instantiation, called twice for a COD->INT->COD implementation of a turbo-coder.

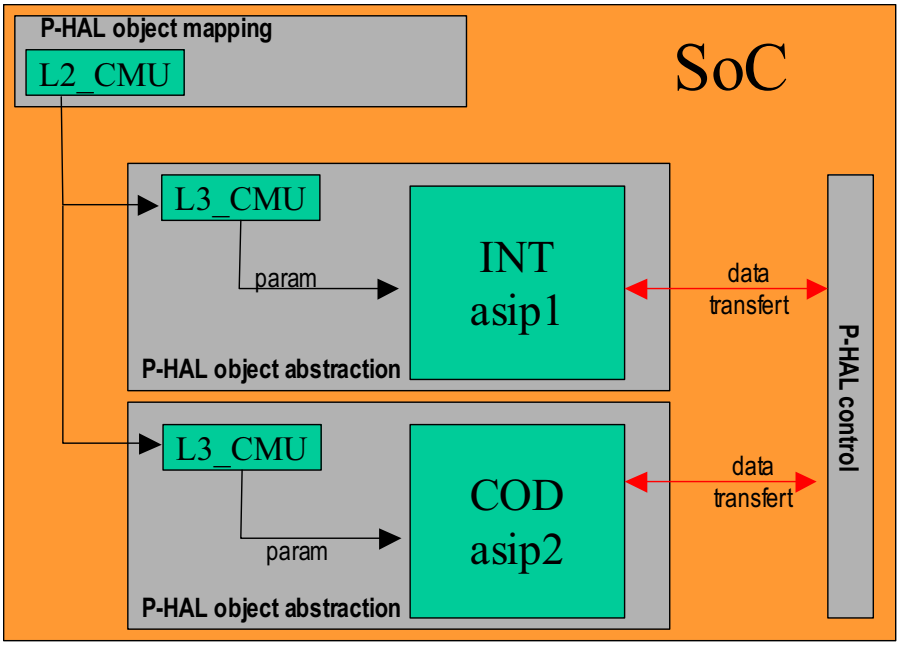


Figure 2-2 – SoC global architecture

The level of configuration assumed in the ASIPs corresponds to a parameter function call change. Each ASIP being designed on a common operator approach, this corresponds to a parameter change of the common operator. The operator implemented by each ASIP can be modified during the execution by a simple change of a global variable (at a pre-defined memory location) referenced by one of the function parameters.

2.1.3 P-HAL FUNCTIONALITIES

The main functionalities that the P-HAL should provide to a reconfigurable system are the following:

- Sequencing of the operations to be executed.
- Data interconnection mechanism between ASIPs
- Change or view the values of operator's variables (a set of them, predefined at design time)
- Statistics and behaviour reporting

Only the first and second functionalities are included in this implementation.

2.1.4 ASIPs or PROCESSING ELEMENTS

The two ASIPs or Processing Elements (PE) considered are described in Figure 2-3 for the interleaver, Figure 2-4 and Figure 2-5 for the convolutive coder. The way the parameterization may be used is also illustrated. Simplicity is kept here to avoid implementation issues and keep main actions on the design philosophy and methodology for reconfigurable systems.

- **Interleaver**

The aim is to change in the output data stream the order of the input data.

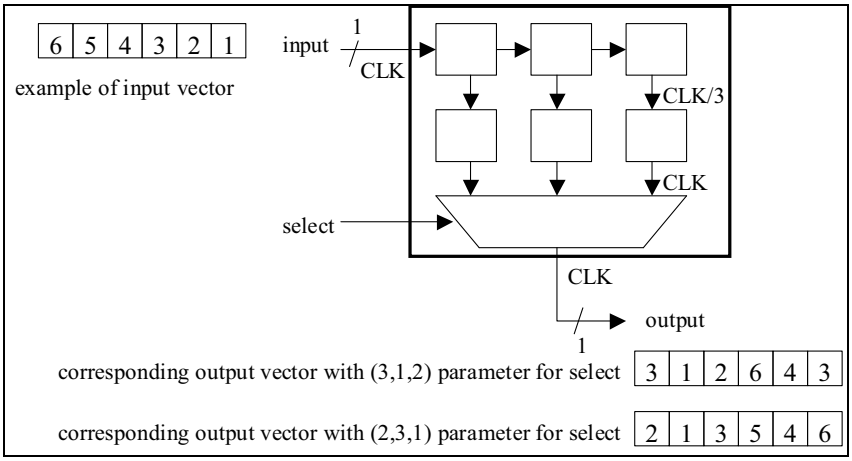


Figure 2-3 – Interleaver detailed structure

- **Coder**

This is a convolutional coding of rate ½. For one input bit, two output bits are generated.

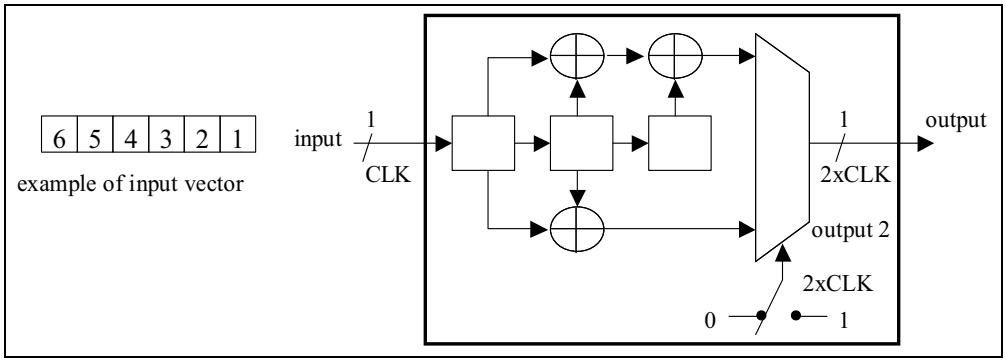


Figure 2-4 – Coder COD(1-1-1, 1-1-0)

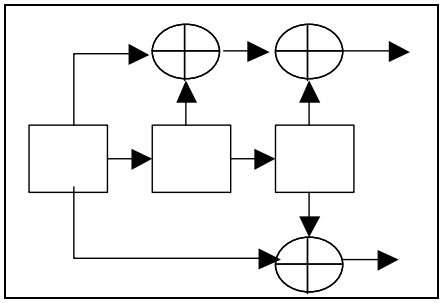


Figure 2-5 – Coder COD(1-1-1, 1-0-1)

The reconfiguration scenarios planned in the scope of this simple demo are only about parameters changing, as it should be in a perfect and ideal common operator approach, which is sufficient to make the complete system reconfigurable in real-time.

2.1.5 RECONFIGURATION SCENARIOS

Incremental scenarios and run-time reconfiguration has been considered.

2.1.5.1 Scenario1: involving only one L3_CMU

In that scenario we changes from one code to another.

- from COD(1-1-1, 1-1-0) to COD(1-1-1, 1-0-1)

- only a parameter change for one ASIP

2.1.5.2 Scenario2: involving L2_CMU and the two L3_CMUs

In that scenario we changes like moving from implementing a CRC to a convolutional coder.

- change both ASIPs functionally at the same time (synchronisation managed by L2_CMU)
- it still remains a parameter change thanks to common operator approach of each ASIP

2.1.5.3 Scenario3:

Use only one COD instantiation and reconfigure it at run-time to do first and, after interleaving, second coding operations

- This is the illustration of the common operator approach as COD is used twice in the same design without a double instantiation.
- P-HAL manages the data exchanges and buffering to support reconfiguration overhead (which is minimal)

2.1.6 METHODOLOGY IMPACTS

P-HAL manages the communications between ASIPs so that they can be forgotten at design time for radio designers. P-HAL also supports the control and the coordination to meet timing constraints and reconfiguration process.

The L2_CMU and L3_CMUs elements are only concerned by dynamic reconfiguration. They are included to make possible the change of the parameters at run-time. They must be created at design time jointly with the common operator that should follow the parameterization approach defined by SUPELEC.

3 IMPLEMENTATION DETAILS

3.1 SoC DESCRIPTION

In order to achieve the identified objectives, several elements must be implemented to work concurrently inside the SoC:

- A set of PE.
- One P-HAL Element for each PE.
- One P-HAL Platform Master.

Each PE is attached to one P-HAL Element and only to one of them. Only the P-HAL is connected to the system network and so we abstract it to the PE. This way, PE doesn't have knowledge about what kind of network is using the SoC, only P-HAL does.

If necessary, a P-HAL Platform Master will do the tasks of routing to an external interface.

In this first approach, the network will be very simple. Each P-HAL element will be connected only to another one. Packets are right shifted through each module. If the destination is not itself packet will be routed to next one, and so on, till packet finds its destination (Figure 3-1).

The last module is the Platform Master and must be connected to the first one. This module, though, will do routing tasks as it would have to output interfaces through witch it will route data packets.

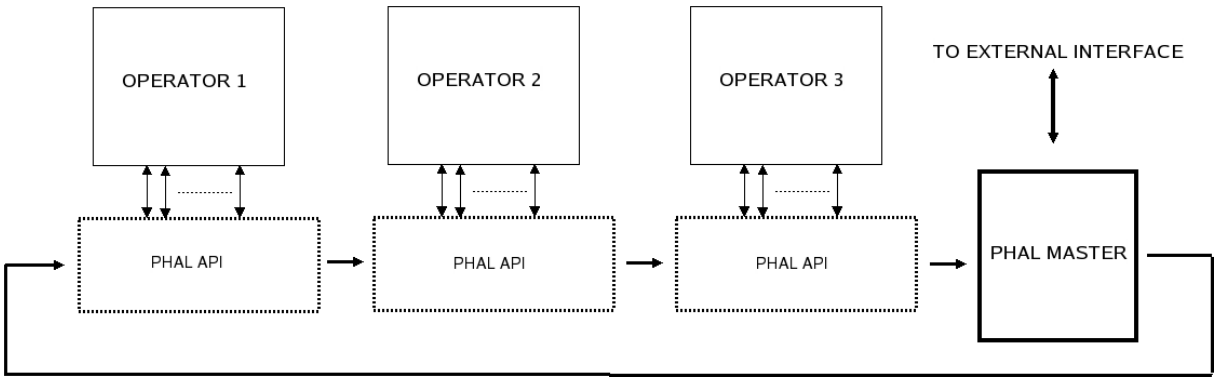


Figure 3-1. SoC Architecture.

3.2 P-HAL implementation details

In the following paragraphs we will present the details about the implementation of P-HAL. First of all we introduce P-HAL and its functionalities and services. Later, we deduce its requirements and show the proposed design and its implementation results.

3.2.1 P-HAL DESCRIPTION

When designing a radio application, there is always a set of operations (or tasks) that must be executed in some kind of scheduling. In order to be executed, these tasks must be mapped into a Processor Element (PE).

Without the PE knowing in which system is going to be implemented or even more, which will be the final application, some kind of Abstraction Layer must be placed between the PE and the system

(Figure 3-2). So, the PE designer only needs to know how to use the services and functionalities that P-HAL provides.

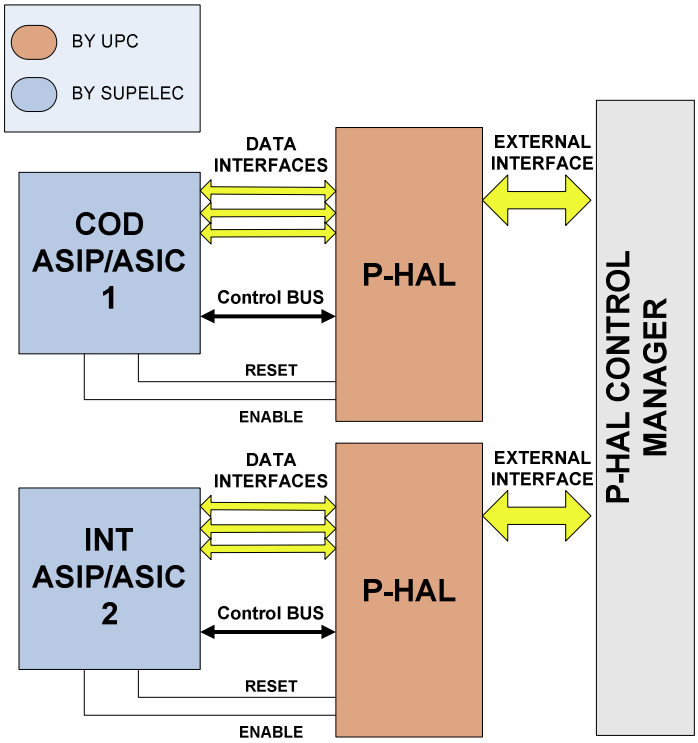


Figure 3-2. Implementation details.

3.2.2 P-HAL API SERVICES

As stated before (section 2.1.3) the services available to the PE are the following (only the implemented services are described):

- Write data to a set of virtual data interfaces (FIFO-like).
- Read data under request from a set of virtual data interfaces (FIFO-like).
- Execution and initialization timing.

In the following paragraphs this services are detailed.

3.2.2.1 Virtual data interfaces

In this first approach, a very simple data interface is going to be implemented consisting in a simple synchronously serial bus. As explained before, P-HAL is abstract enough to permit easy future improvements in its data interfaces.

The data is synchronized with a **bus_clk** generated by P-HAL but for simplicity the **bus_clk** will be the same as the global system clock.

The data scheme is the same for both input and output interfaces, the only difference is in the protocol, as reading is under request.

A signal summary and naming convention is described in section 3.2.2.1.3.

3.2.2.1.1 Write data to an interface

First of all, there is a common signal shared for all interfaces (input and output):

- 1-bit signal for **BUS CLK** *(bus_clk)*. Input from the module.

Then, for each interface we have the following signals:

- 1-bit signal for **data sending** *(data_out_i)*. Output from the module.

The format of the data is detailed in section 3.2.2.1.4).

From the PE view, data interfaces are FIFO-like, so it can write data at any time. In future implementations, buffer state will be reported through the control interface.

3.2.2.1.2 Read data from an interface

The signals available for each reading interface are:

- 1-bit signal for **data receiving** (*data_in_i*). Input from the module.
- 1-bit signal for **data requesting** *(data_req_i)*. Output from the module

In this case, data is **requested** by the PE. The procedure is the following:

- Write '1' to *data_req_i* signal
- Wait one period of *bus_clk* signal.
- Write '0' to *data_req_i*.
- Data will be received through the *data_in_i* signal following a predetermined format (section 3.2.2.1.4)
- If no data is available nothing will be send.

3.2.2.1.3 Signals summary

Naming conventions:

N means number of writing interfaces.
L means number of reading interfaces.

Signal	**Name**
Bus clock	bus_clk
Writing data signal	data_out_0, data_out_1, … data_out_*i*, …. data_out_N
Reading data signal	data_in_0, data_in_1, …. data_in_*i*, …. data_in_L
Data Requesting signal	data_req_0, data_req_1, … data_req_i, … data_req_L

3.2.2.1.4 Data Format

The format of the data is a very simple serial scheme. It is the same for input and for output interfaces.

Data is aligned to 32-bits words. It is written or read from the interfaces as:

- 1 start bit. Signals a '1'.
- 32 bits (data word).
- After these 33 bits, if a 0 is received transmission is ended. Otherwise, if a '1' is received another word will be processed.

All the bits (data and start bit) have to be read or written synchronously with the *bus_clk*.

3.2.2.2 Execution control

The execution of the PE will be ordered by the Enable/Disable signal. There is only 1 wire signal named *enable*. The operation is as follows:

- Signal to '1' means ENABLE.
- Signal to '0' means DISABLE.

There is also a reset signal that indicates the PE to re-initialize itself:

- Signal to '1' means NORMAL OPERATION:
- Signal to '0' means RESET

3.2.3 GUIDELINES FOR DEVELOPING A PE UNDER P-HAL API

When developing an operator (or PE) to be embedded in an SoC environment using P-HAL API some considerations must be taken into account:

- Must follow interfaces data scheme.
- Application-level control must be managed by an application operator (same one or another) using normal data interfaces.

3.2.4 P-HAL ARCHITECTURE DETAILS

In the following section, the details about the ASIP design are described. The design is inherited from the objective specifications, which are specified first of all.

3.2.4.1 Design objectives

PHAL specifications determine a set of objective which will, at the end, require one or other processor/architecture type. These are mainly as follows:

- *Control oriented*: PHAL is not intended for hard data computing tasks, otherwise, it will analyse data and do some actions depending on it.
- *Complex data structures*: Following last one, PHAL will manage data and control packets and take decisions on that. So, an easy and flexible way to access this kind of data is required.
- *Small*: The design must be as small as possible. This is because PHAL is going to be a support entity, so we don't want to overhead the final SoC in size.
- *Real time I/O data access*: Finally, the design must be able to access to a set of simple input and output serial interfaces in real time.

As a consequence of all these objectives, we determine the following set of requirements:

- *Control oriented* ➔ We need a complex state machine, that is, a processor.
- *Complex data structures* ➔ Programs can not be written with assembly instructions so a processor supporting a C Compiler is required.
- *Small, specific IO logic* ➔ Special customization of processor architecture.

Observing the requirements of the design, we can finally conclude that an ASIP is the optimal solution. This architecture provide us flexibility enough to design a processor as small as we require (or as we can), embedding the special logic functionality and providing support for common C compiled instructions.

The design will be a Reduced Instruction Set Computer (RISC) with further customizations to provide the required functionalities.

It is important to notice that although a programmable device was not one of the requirements (at a first chance, PHAL is not needed to be reconfigured) it has been found that an ASIP is the best solution to meet our needs, providing an extra functionality at a low-cost increment.

3.2.4.2 Architecture overview

Following the previous requirements, for the design of the processor we started from a 3-stage pipeline RISC processor and do some modifications on it. According to the small size requirement, the first modification was to reduce the pipeline to a **2-stage** one:

- First stage is the fetching stage. Branch and interrupt triggering (see chapter 3.2.4.5) are also included.
- Second one is for decoding and execution.

This pipeline reduction reduces the global frequency clock rate but we must consider size as more restrictive than speed.

Also, there are no needs to manage big amounts of data at same time, so, the GPR has been reduced to **16 registers** (32-bit wide). Memory accesses are also **word aligned** (32 bits) in order to reduce global complexity.

Some other special logic (detailed in chapter 3.2.4.4) to deal with external serial interfaces is embedded in the processor design, working apart of it, but concurrently with it. Finally, a set of special instructions are embedded in the processor architecture in order to access this IO data. An overview of the processor architecture can be seen in next figure..

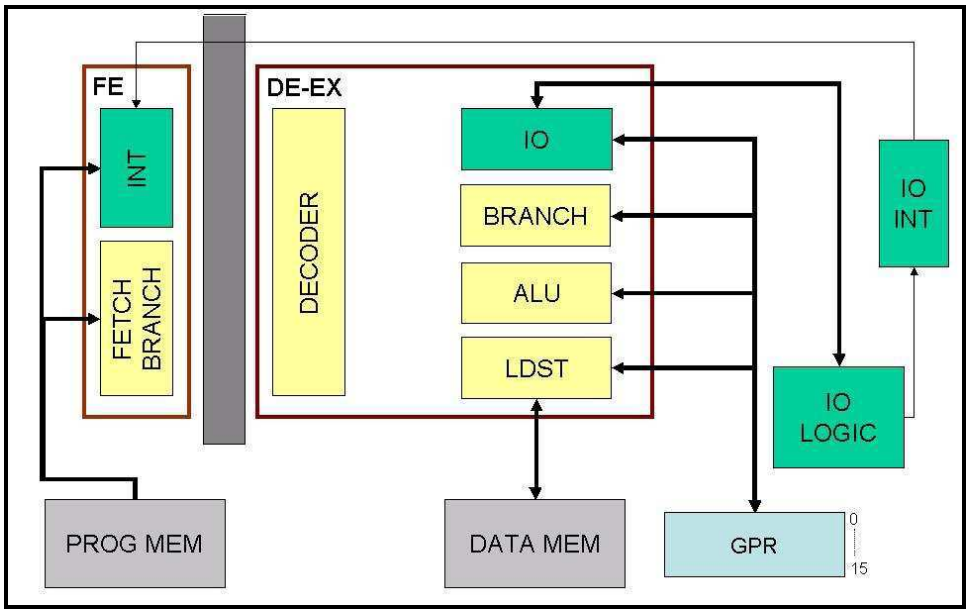


Figure 3-3. RISK processor overview.

3.2.4.3 High level architecture. Mixed C and ASM code execution

If we analyse more deeply the requirements of the processor from the view of its final application (see previous chapters), we notice as we will have to deal with two types of scenarios. Next, we can see them and its consequence in means of system requirements:

- Data arrives or is requested and an **immediate** “real time” reaction is needed. Speed *is* important.

- Packet/object is generated and a heavy and **complex** processing/analysing is required. Speed *is not* important.

As a consequence of these two requirements, we design the *software* architecture as follows:

- *ASM code* is called by an interrupt event when data arrives or requested in order to process it as quick as possible. The assembler code uses the specific instruction *GET* to save data or uses a *PUT* instruction to send data (instructions detailed below).
- *C compiled code* is used during the normal execution. It continuously checks for new packet/object to process and does it when generated.

This scenario is represented in Figure 3-4.

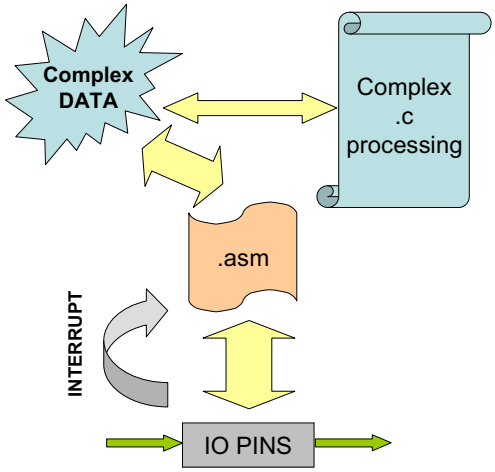


Figure 3-4. Code Development.

In the following paragraph we describe the instructions implemented in the processor.

The **special instructions** are intended to access the IO interface registers (located outside the GPR) for reading and for writing. These are the implemented instructions (see chapter 3.2.4.4 for details on the interfaces):

- GET: copies the input register value to a GPR for later processing using common instructions.
- PUT: copies the value from a GPR to the output register in order to send it.
- SETPIN: Sets the output pins enable or reset to a certain value.

In the other hand, a set of **common instructions** are included in order to support C Compiler. These are summarized next:

- *Data management*: ldw, stw, ldc, ldi, etc.
- *Branch control*: conditional: bne, blt, bgt, ble, etc.; direct: jump; interrupt return: bIRP (see chapter 3.2.4.5)
- *Arithmetic*: add, sub with registers and addi with an immediate value.
- *Logic*: or, xor, and. And with an immediate: ori, xori, andi
- *Shifters*: left and right logic and arithmetic shifters with register and immediate value.

3.2.4.4 IO embedded logic

According to the data interface specifications (see 3.2.2.1) we designed a small logic circuit to manage each of them. There are two kind of interfaces, input and output ones which are described in the following paragraphs. Each interface has each own circuit.

INPUT INTERFACE (Figure 3-5):

- Wait's to receive the start bit (input_data='1') indicator.
- Then a counter starts counting (from 0 to 31, or other bit wide).
- A demux and a set of OR gates saves the serial data flow into register REG0 at BUS_CLK rate. The counter indicates the position of the bit.
- When counter finishes REG0 data is copied to REG1 and interrupt is flagged.
- That will call an assembler routine which will read REG1 data using the *GET* instruction.
- The system is ready for new data. It will be saved to REG0 while the processor reads REG1.

Notice the timing constraint in the assembler routine. It must fast enough to save REG1 data before a 32 bit data is generated again at bus clock rate. In the worst case, when CPU_CLK and BUS_CLK are the same, assembler instruction must be smaller than 32 instructions.

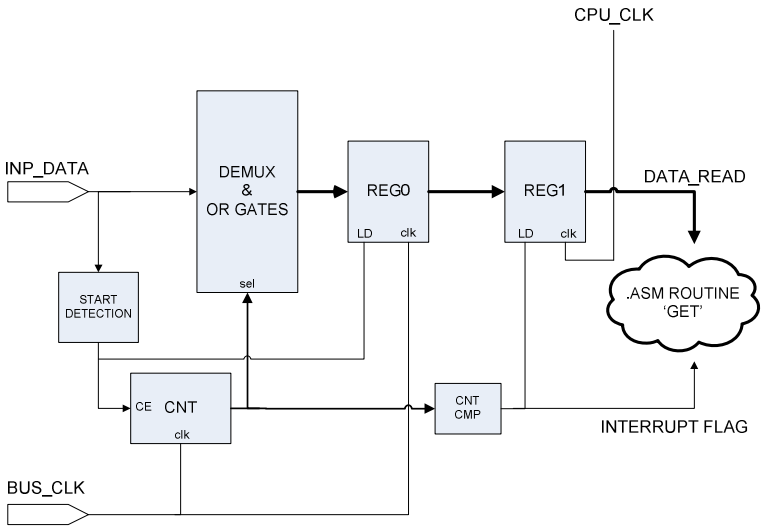


Figure 3-5. Input Interface.

OUTPUT INTERFACE (Figure 3-6)**:**

- Data is requested putting a '1' in *data_requestN* signal where *N* indicates interface number.
- Then an interrupt is flagged and an assembler routine executed.
- The routine uses the *PUT* instruction to load the data to REG0 register.
- The start bit is send and counter starts running.
- As counter goes from 0 to 31, bits are read, one by one, from REG0 register until the end.

Here the timing constraints are self protected by the *PUT* instruction. Before writing again to REG0 register, it checks the counter state and only writes if no data is being send. That implies that the assembler routine must check the *PUT* return state to ensure that data has been send.

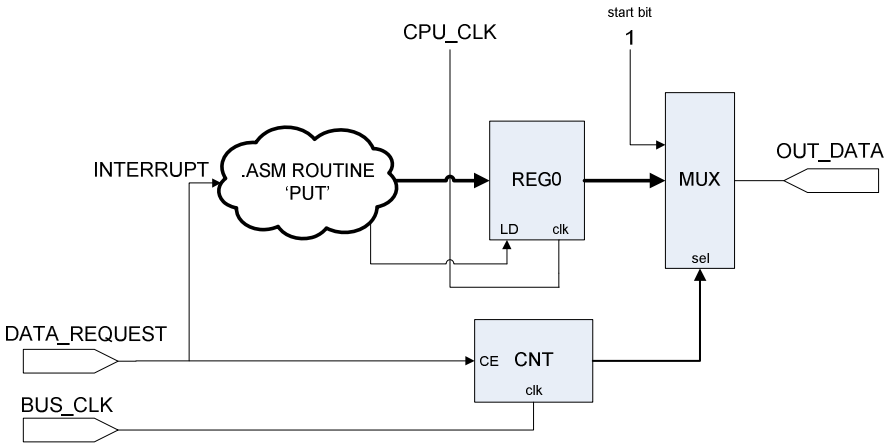


Figure 3-6. Ouput interface

3.2.4.5 Interrupt handling

We have seen how the processor can be stopped at any time of its execution and be required to run an assembler routine. This is accomplished using interrupts.

The interrupt handling and triggering logic is located in the Fetch stage of the pipeline. As the routines to be executed are known and fixed, we didn't implement an IST (Interrupt Service Table) and used fixed interrupt destinations. In the following paragraph we describe how we implemented the interrupt handling:

- Each IO interface has its own logic, then its own interrupt line. This is named *inp_intN* for input interfaces or *data_requestN* for output interfaces. *N* indicates interface number.
- Every interrupt line is attached to a register.
- At **Fetch stage** processor checks for interrupt registers state and executes the branch to the known address. The priority is as follows:
  - First check *inp_int0 … inp_intN.*
  - Then check *data_request0 … data_requestN.*
  - Then check for an interrupt return state (*bIRP)*.
  - Finally check branch request register state.
- When the branch occurs, the program address is saved into the Interrupt Return Point register.
- Now the interrupt routine is executed.
- When finished, it calls the *bIRP* (Branch Interrupt Return Point) which will set the program counter to the previously saved IRP register value.

3.3 PROCESSING ELEMENTS (PE) IMPLEMENTATION DETAILS

3.3.1 OBJECTIVE

First objective of the Processing Elements (PEs) is to perform the processing in real-time but maintaining the capability to perform a reconfiguration during the execution time. To be capable to manage such situation the P-HAL controls and schedules the data flow between each PE, and manage its reconfiguration by changing some parameters. Processing Element's functionality has been be developed in plenty C code, and compiled and implemented on a RISK architecture design with LISATeK tool.

3.3.2 PE DESCRIPTION

3.3.2.1 Interleaver description

Figure 3-7 is a functional electronics-oriented block diagram of the interleaver PE.

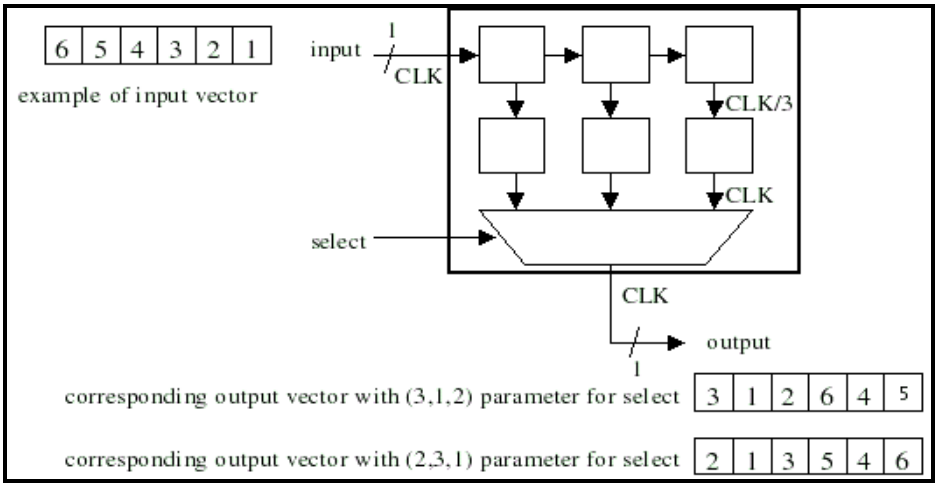


Figure 3-7. Interleaver

Here the key point is that the parameterisation is done by re ordering the output data in the way required by the radio application.

3.3.2.2 Coder description

Figure 3-8 and Figure 3-9 are a functional electronics-oriented block diagram of the coder PE with 2 different configurations or set of parameters.

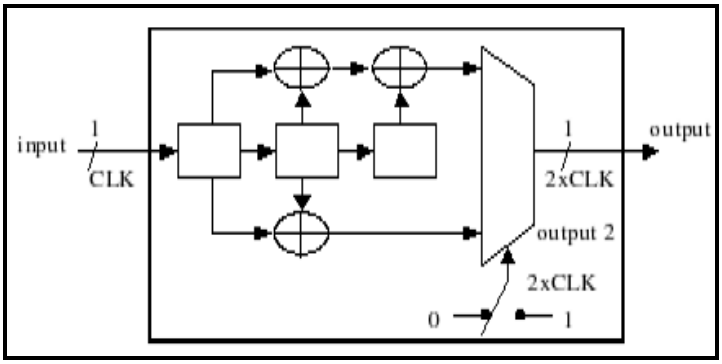


Figure 3-8. Coder COD (1-1-1, 1-1-0).

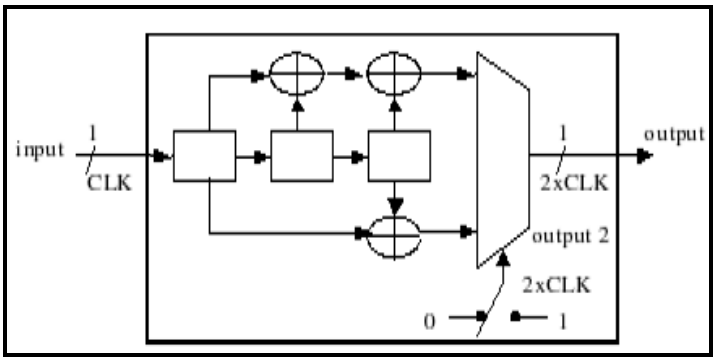


Figure 3-9. Coder COD(1-1-1, 1-0-1).

This is a convolutional coding of rate ½. For one input bit, two output bits are generated.

3.3.3 PE FUNCTIONALITIES

3.3.3.1 Coder functionalities

This PE has to store bits in order to arrange them in a pre order polynomial expression. With our approach this convolutional code can be modified in execution. To perform such a thing, a Configuration Management Unit L3_CMU manages the PE and makes parameterisation under the request of L2_CMU. Figure 5 shows a schematically view of Coder.

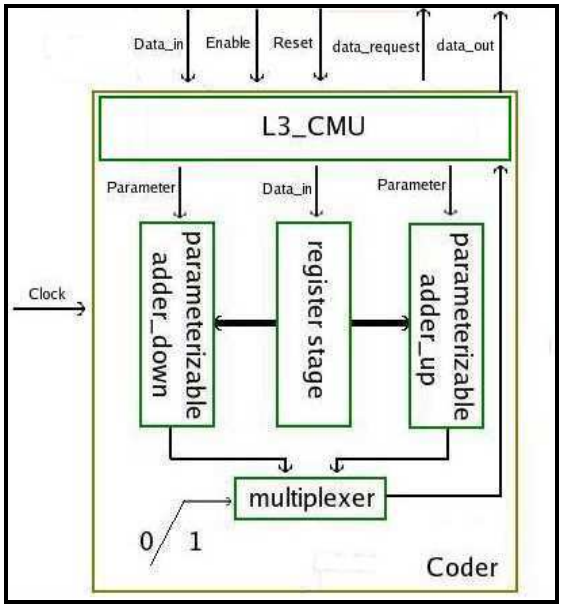


Figure 3-10. Coder.

The input data for this PE are allocated by P-HAL and are distributed through FIFO-like interface in the Coder. Each cycle clock, three bits of the 32 bits data_in register are shifted in three registers toward the most significant bit. Then according to parameters, data are processing trough PE. At each clock cycle, the output of Coder takes alternatively the value of output Adder_up and output Adder_down. When a bitstream of 32 bits is ready, the first output packet is generated in L3_CMU and sends to P-HAL. Then after 32 clock cycles, the second packet is generated and sent.

Data format send by P-HAL to coder is:

Bit 31 : detection	*Bit 30: no use*	*Bit 29...8*	*Bit 7...4*	*Bit 3...0*
If (1 : parameter)	No use		Parameter for adder_up path	Parameter for adder_down path
If (0 : data)	No use	Data bitstream		

Table 1. Coder data format.

Parameter value can be:

Parameter value	*Operation*
0x0	Output Adder_up = 0
0x1	Output Adder_up = register0
0x2	Output Adder_up = register1
0x3	Output Adder_up = register2
0x4	Output Adder_up = register0 Xor register1
0x5	Output Adder_up = register0 Xor register2
0x6	Output Adder_up = register1 Xor register2
0x7	Output Adder_up = register0 Xor register1 Xor register 2

Table 2. Coder parameters.

The same set of parameter value is used for the Output Adder_down.

3.3.3.2 Interleaver functionalities

An interleaver changes the order of input data to protect the transmission against burst errors. Our PE uses a set of registers making in our case 3 column and 2 lines as shown in Figure 3-11. Here the L3_CMU parameterizes the output data order. Figure 3-11 shows a schematically view of this PE.

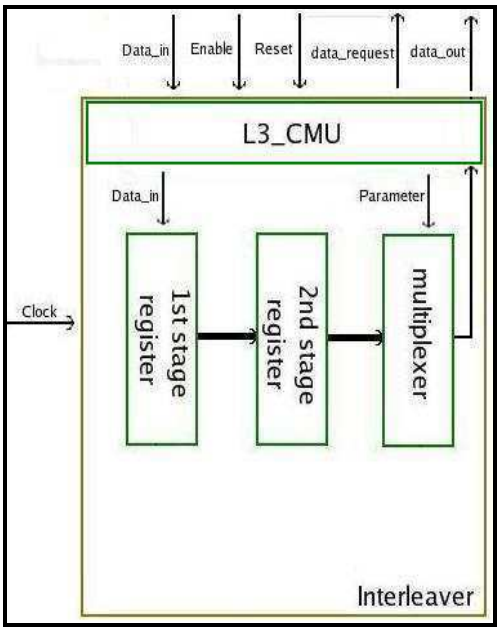


Figure 3-11. Interleaver.

The first stage register refers to the first line of the register set, same for the second stage. The input data for this PE are allocated by P-HAL and are distributed thought FIFO-like interface in the Interleaver. Each cycle clock, data are shifted in the registers toward the most significant bit. Data goes through the first stage register and when this first line is full (i.e. three data bits are stored) the second stage immediately loads the 3 values in its three registers. This can assume a continuously data flow. Indeed, input data are interleaved by packet of three bits in the second stage register since three other bits are load in the first stage register. When the second stage register is loaded, output data are re ordered with the help of multiplexer and in the order set by the L3_CMU.

Data format send by P-HAL to interleaver is:

Bit 31 : detection	***Bit 30: no use***	***Bit 29...8***	***Bit 3...0***
If (1 : parameter)	No use		Parameter for multiplexer
If 0 : (data)	No use	Data bitstream	

Table 3. Interleaver data format.

Parameter value can be:

Parameter value	***Register order output***
0x0	(register0, register1,register2)
0x1	(register0, register2,register1)
0x2	(register1, register0,register2)
0x3	(register1, register2,register0)
0x4	(register2, register0,register1)
0x5	(register2, register1,register0)

Table 4. Interleaver parameters.

3.4 CONFIGURATION MANAGEMENT UNITS (CMU): IMPLEMENTATION DETAILS

3.4.1 OBJECTIVES

To make easy the reconfiguration of the Processing Element at run time, we define Configuration Management Units (CMUs). Those CMUs can be distributed on a heterogeneous HW design, GPPs, DSPs as well as FPGAs.

In a complex design suitable for flexible radio, we define three management levels of abstraction to manage the flexibility of the design and support different configurations it has to run (Figure 3-12). The higher level of abstraction is L1_CM. This level has a global knowledge of the design; it has no idea of the PE repartition neither through the design nor from the technology target (GPPs, DSPs or FPGAs). L1_CM is the chief manager, its roles are platform initialisation and configuration change. At level2 and 3 there are one or several Configuration Manager Units. L2_CMU is a configuration management at the scale of a HW component that runs several processing elements. Those units manage the configuration of these processing elements sent by L1_CM. L3_CMU is a configuration unit associated with each processing element. It manages the execution and the configuration of every processing element.

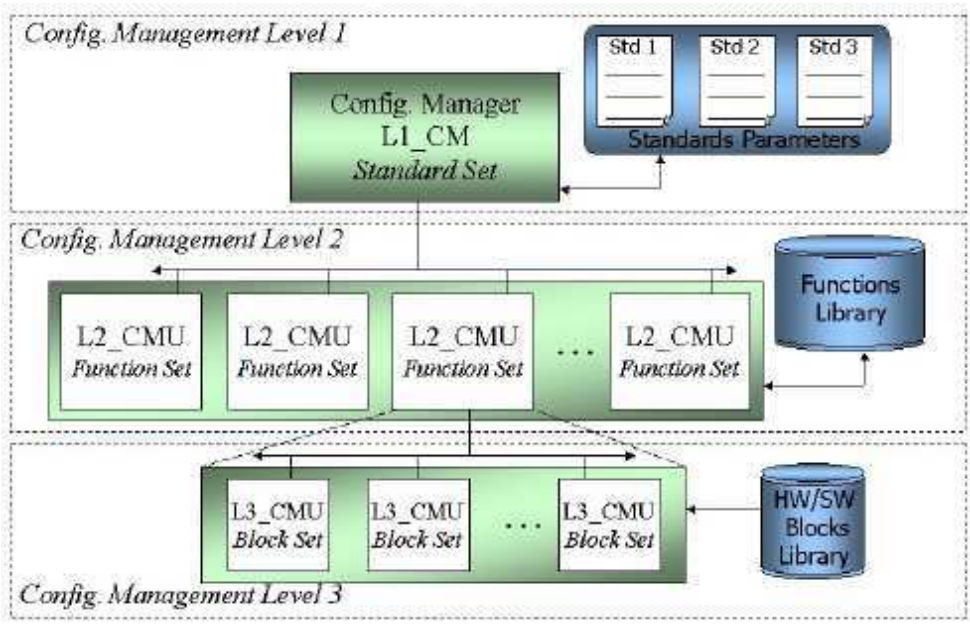


Figure 3-12. Hierarchical management.

In this joint design, L1_CM will be written in plenty C code. The P-HAL, that manages parameters through the design, can be considered as a L2_CMU. L3_CMUs will be integrated in PE design.

1.3 Configuration Management unit of PE

For each PE, we have one L3_CMU. Those CMUs performs:

- Check enable and reset signals,
- Unpacket data sent by P-HAL,
- Check if input is a parameter (or a set of parameters) or data,
  - if parameter : make a reconfiguration of PE,
  - if data : make them accessible for PE,
- Packetize output data and send them to P-HAL.

3.4.2 WORKING PROCESS

3.4.2.1 Initial phase

At starting time, L3_CMU waits for their PE to be enabled. When enable take value “true”, a data_request signal is sent to P-HAL and L3_CMU waits for a start bit. After a 100 clock cycle if there is no start bit, L3_CMU sends a new request signal.
When a bitstream of 32 bits has been loaded in PE internal register, a flag takes the value “true”. Then L3_CMU checks the value of the MSB. According to that value, PE is configured (MSB = 1) or data are processed (MSB = 0). When it’s done, a data out start bit is sent to P-HAL followed by a 32 bits bitstream. Then if enable is “true”, a new data_request signal is sent to P-HAL.

3.4.2.2 Running time.

Figure 3-13 shows the interfacing between the PE and the P-HAL. As both interfaces PE/P-HAL use the same rules set we will take here the example of Coder/P-HAL. As explain before, the P-HAL get packet and check if data is for its PE or not. If data have reached the correct destination, P-HAL extracts data and put them in a virtual interface for the use of its Processing Element. If not, data are sending to an external FIFO for the use of another P-HAL. After being enabled, If PE needs data it sends a request to it P-HAL then if there are data, operator access to them otherwise it has to wait.

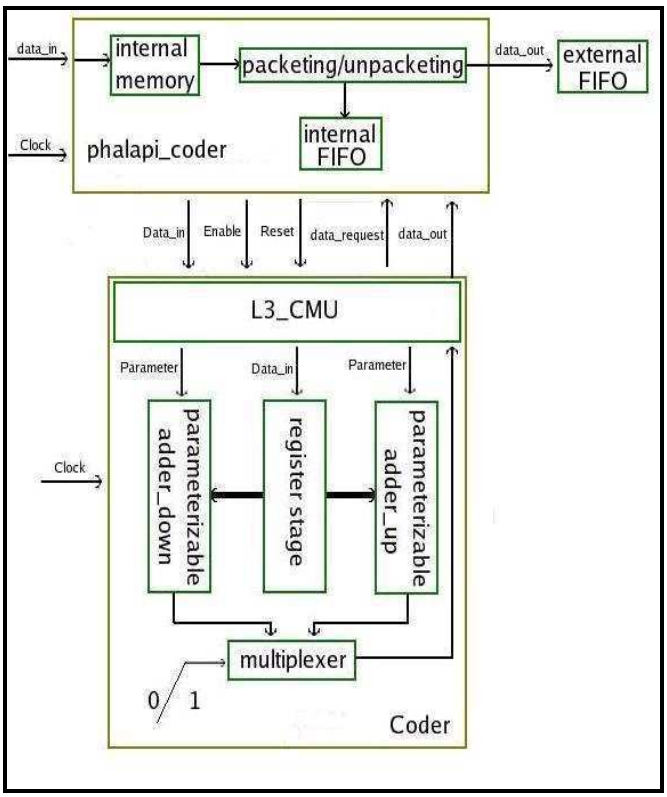


Figure 3-13. PE and PHAL

The data transfer is set between PE and P-HAL according to P-HAL protocol:

- Read data under request (data protocol is 1 in_start bit + 32 bits),
- Write data in P-HAL (data protocol is 1 out_start bit + 32 bits),
- Check enable and reset signal.

After a data request signal, a bitstream word is sent to PE. Then L3_CMU checks if this is parameters or data. This is done according to the value of the MSB. If the value is 1 then parameters are detected, if the MSB value is 0 those are data. Then, when configuration is done according to parameters value or when data have been processed, if PE is enabling, it sends another data_request to PHAL and waits for data.

4 OVERALL RESULTS: JOINT DESIGN

4.1 Initial Objective

In this joint project, the initial objective was to merge partners experience in a joint design for the use of flexible radio. Indeed we have three partners with three points of view on flexible design: SUPELEC, from France, with the common operator and reconfiguration management point of view, UPC, from Spain, with the P-HAL adding flexibility in communication and task scheduling and TUA, from Germany, with ASIPs and system level design. The experience aimed for defining a methodology for flexible radio design.

4.2 Results: Processor architecture

Both processors for the corresponding PEs have quite similar architecture. The initial P-HAL architecture, which was design on summer by UPC, is generic enough to permit that. Processors are based on RISK architecture with some special IO instructions in order to manage the data quickly, embedded logic, two pipeline stages and 15 GPRs. Figure 4-1 gives an overview of the processor architecture.

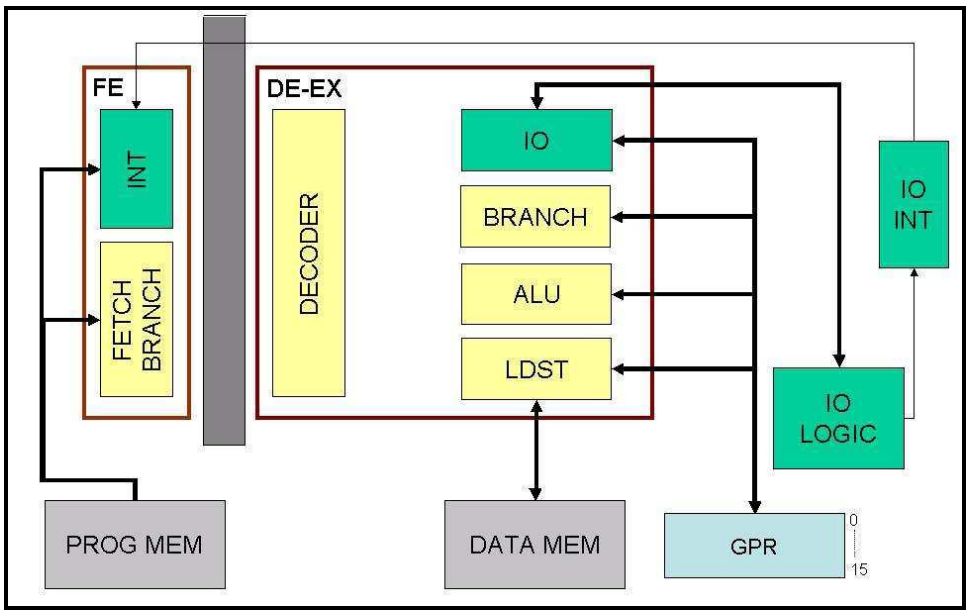


Figure 4-1. Processor architecture.

4.3 Results: Processors Integration

For the simulation, 4 processors are integrated and synchronised between each others. This part is done by a P-HAL MANAGER (acting as L1_CM) written in plenty C code which instantiated and synchronised four ASIPs. By the use of LiSATool debuggers we can make a HW exploration (registers values, pipeline stage, memory address …) of processors at run time. In order to send data to PHAL master, a user interface write in plenty C code is used.
For this design we have integrated a P-HAL master receiving user commands, a P-HAL acting as slave to P-HAL master, a Coder and an Interleaver. P-HAL master manages data in the design. Figure 4-2 shows how it looks like.

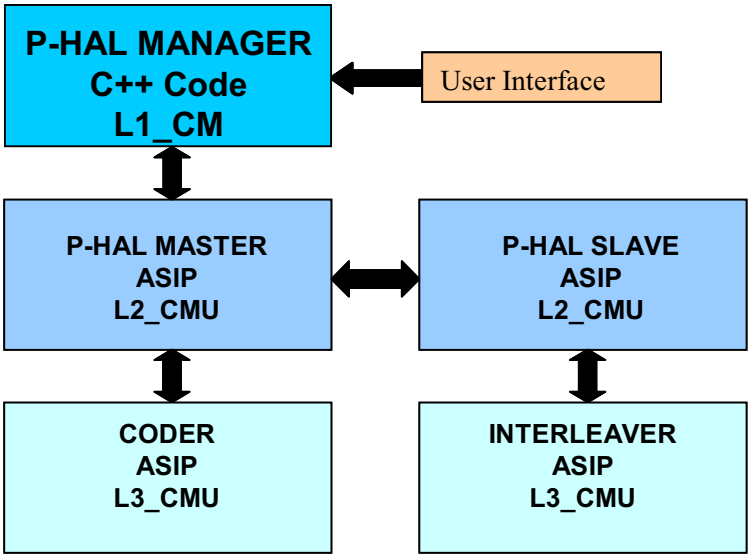


Figure 4-2. Joint Design Structure.

Results of the execution can be seen in the P-HAL MANAGER window. In order to give a high level SW exploration, a GUI interface as been created. So we can see in real time the data flow through the design. Figure 4-3 give an overview of this graphical interface.

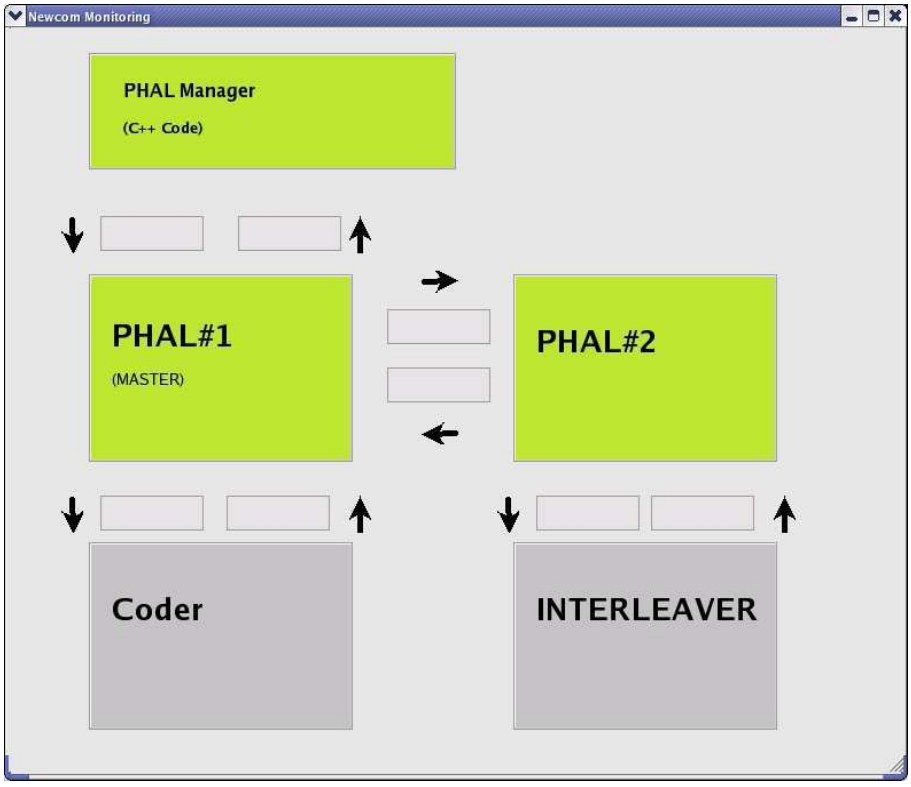


Figure 4-3. GUI interface.

From this interface, we can see the packets sending by P-HAL Manager to Master and Slave and see hexadecimal bitstream word input (output) sent to (by the) PE by clicking on the rectangular shapes near to the arrows. Also PE change colour if disable (grey) or enable (green).

4.4 Results: Execution

4.4.1 Two executions mode

With this design we can run in two execution mode. One is a (COD ► INT) execution which is calling Mode1 and another one Mode2 (COD ► INT ► COD) with a re-use of the Coder ASIP. Both modes give access to reconfiguration via parameters change. For configuring the design in one mode two bitstreams packets are sent, one to P-HAL Master and the other one to P-HAL Slave. Those bitstreams configure the interface P-HAL/PE. For example packet can configure the P-HAL Master in the way that data processed by the Coder must be processed by the Interleaver after or be managed to the external output (P-HAL Manager in our case). In case of Mode2 execution, a third packet has to be sent to P-HAL Master in order to avoid infinite loop between the Coder and the Interleaver. This is the duty of the P-HAL Manager to make this configuration in execution (before the second pass in the Coder).

4.4.2 Execution step

Once user has selected the execution mode, he has to initialise both Processor Elements with a parameter. To perform this, a hexadecimal word bitstream is sent to each PE via a packet containing the PE destination and the parameter. As PE makes the difference between data or parameters on the MSB, parameter sent to Coder must be like 800000XX and for Interleaver 8000000X where X is the value of the parameter (X=[0..7] for Coder and X=[0..5] for Interleaver). See Table 1 and Table 2for parameters impact on PE. After parameter is sent, we can send data (note here that as bit 31 is used for checking if this is a parameter or a data and that the bit 30 is not used, hexadecimal bitstream data can take the maximum value 3fffffff).

At the beginning of an execution, both PE are not initialised and not enable. We have tow kind of execution. In Mode1 we can run PE in concurrent mode, meaning both of them can stay enable at the same time. This is not true actually for Mode2, some work have to be done on this. To resolve this, we can add one input interface in Coder ASIP for the output data of the Interleaver, or add special scheduling in order to avoid mixing data in Coder for a continuous data flow. We can run in concurrent mode with the help of circular buffer in P-HAL that store data, waiting for request signal from PE.

The protocol for Mode1 execution is the following:

- Enable Coder and Interleaver PE
- Initialise them with a parameter
- Send hexadecimal bitstream data to Coder
- Look at the result

The protocol for Mode2 execution is the following:

- Enable one PE at a time (other one always stay disable)
- Initialise enable PE with a parameter (one at a time)
- Send hexadecimal bitstream data to Coder (which is enable)
- Disable Coder
- Enable Interleaver, let it process the data and disable it
- Disable Interleaver
- At this point, P-HAL Manager sends a reconfiguration order for P-HAL Master/PE interface in order to avoid the infinite loop.
- Enable Coder
- Look at the result

5 CONCLUSIONS

A design that integrates 4 RISKs processors as been creates. A simple (**coder ► interleaver**) execution can be done, as well as a reconfiguration to make a (**coder ► interleaver ► coder**) one. The joint design execution can be reconfigured with simple instructions, and the behaviour of PE coder and interleaver can also be changed at run time. Some future work will be to design a specific ASIP for interleaving and coding task. Indeed at the moment Processing Elements are very simple and can not do complex execution. Indeed focus was not on PE design but on how make all work together. But thanks to ASIPs the re-programming is quite easy.

6 REFERENCES

[1] J.-P. Delahaye, J. Palicot, P. Leray, "A Hierarchical Modeling Approach in Software Defined Radio System Design," SIPS 2005, Athens-Greece, Nov. 2005

[2] Jean-Philippe DELAHAYE, Pierre LERAY, Christophe MOY, Jacques PALICOT, "Managing Dynamic Partial Reconfiguration on Actual Heterogeneous Platform", SDR Forum Technical Conference, Anaheim (USA), November 2005

[3] Loïg Godard, Christophe Moy, Jacques Palicot, "From a Configuration Management to a Cognitive Radio Management of SDR Systems", *CrownCom'06*, 8-10 June 2006, Mykonos, Greece

[4] Xavier Revés, Antoni Gelonch, Vuk Marojevic, Ramon Ferrús "Software Radios: Unifying the Reconfiguration Process over Heterogeneous Platforms",. EURASIP Journal on Applied Signal Processing, Volume 2005, Issue 16, September 2005

Liste des tableaux

Table des figures

Contributions de l'auteur

REVUE

Loïg Godard, Christophe Moy, Jacques Palicot, "An Executable Metamodel of a Hierarchical and Distributed Architecture Management of Cognitive Radio Equipments". Article de revue accepté aux annales des télécommunications, pour le Special Issue sur la Radio Cognitive, publication courant 2009.

PUBLICATIONS INTERNATIONALES AVEC ACTES

- Loïg Godard, Christophe Moy, Jacques Palicot, "A simulator for the design of the management architecture of cognitive radio equipments", 5th Karlsruhe Workshop on Software Radios, WSR'08, Karlsruhe, Germany, March 2008
- Loïg Godard, Hongzhi Wang, Christophe Moy, Pierre Leray, "Common Operators Design on Dynamically Reconfigurable Hardware for SDR Systems", SDR Forum Technical Conference, Denver, USA, 5-9 novembre 2007
- Jean-Philippe Delahaye, Pierre Leray, Loïg Godard, Amor Nafkha, Christophe Moy, "Designing a Reconfigurable Processing Datapath for SDR Over Heterogeneous Reconfigurable Platforms", SDR Forum Technical Conference, Denver, USA, 5-9 novembre 2007
- Loïg Godard, Christophe Moy, Jacques Palicot, "From a Configuration Management to a Cognitive Radio Management of SDR Systems" CrownCom'06, 8-10 June 2006, Mykonos, Greece, pp. 11-15 (first Conference on Cognitive Radio Oriented Wireless Networks and COMmunications)

PUBLICATIONS NATIONALES AVEC ACTES

- Loïg Godard, "Architecture Hiérarchique Distribuée adaptée à la Cognitive Radio", JNRDM 2008, Bordeaux, France
- Loïg Godard, "Méthodologie de conception pour système de radio communication *intelligent*", JNRDM 2007, Lille, France
- Loïg Godard, "Gestionnaire de reconfiguration *intelligent* pour la cognitive radio", JNRDM 2006, Rennes France

COMMUNICATIONS SANS ACTES

- Loïg Godard, Christophe Moy, “A Simulator for cognitive radio equipments”, Kermeta Days 07,décembre 2007 Rennes France
- Loïg Godard, “Concept de design appliqué au domaine de la flexible radio dans le cadre du Réseau d’Excellence NEWCOM", Séminaire SCEE, Juin 2007, Rennes France

RAPPORT TECHNIQUE

Participation à la rédaction du délivrable proposé en Annexe 2 : “Report on HW/SW architecture of the Flexible Digital Platform" dans le cadre de NEWCOM.

Présentation de la démonstration : Flexible MPSoC Radio Design by Programmable Hardware Abstraction and Configuration Management, Institutions : Supélec, Politecnic University of Catalunya, Technical University of Aachen, NEWCOM DISSEMINATION DAY - 15 February 2007, Paris France

CONTRIBUTION AU DÉROULEMENT DES JNRDMS 2006 À Rennes

La conférence des JNRDM (Journées Nationales du Réseau Doctorale en Micro-électronique) est une conférence annuelle organisé en France par les doctorants pour les doctorants du domaine de la micro-électronique. Cette conférence s’est déroulée à Rennes en 2006 sur le campus de Beaulieu. Cette conférence s’organise grâce à l’implication de doctorants dans la recherche de fond pour subventionner l’évènnement (transport, hébergement, inscriptions offertes pour les doctorants y participant), ainsi que de toute la logistique nécessaire au bon déroulement de la conférence. J’ai profité du fait que Rennes soit choisie pour acceuillir la conférence afin de contribuer avec le comité d’organisation présidé par M. Ludovic Barrandon à différentes tâches comme la participation à la création du site internet, l’acceuil des participants ainsi qu’à la mise à disposition de différentes commoditées.

CONTRIBUTION AU CONSEIL DES DOCTORANTS

J’ai participé en tant que représentant SUPELEC au conseil des doctorants de l’école doctorale (ED) MATISSE et à l’élaboration de la première journée des doctorants de

l'IETR en 2007. Le but de cette journée est de permettre à tous les doctorants, de première et deuxième année principalement, de MATISSE de présenter leur travaux de recherche. Cette journée était organiser de façon à ce que tous puissent y accéder (encadrant et personnes extérieures). Il a été remis à l'ensemble des visiteurs un recueil d'acte des participants. L'ED MATISSE étant répartie sur trois campus différents (Beaulieu, INSA et SUPELEC), cet évènement a permis une meilleure visibilité sur les différents travaux en court entre doctorant ne partageant pas un même environnement de travail.

Bibliographie

[1] sharedspectrum. http ://www.sharedspectrum.com/measurements/.

[2] J. Mitola and G. Maguire. Cognitive radio : making software radios more personal. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 6(4) :13–18, 1999.

[3] C. Roland and J. Palicot. A self adaptive universal receiver. *Annals of telecom*, 57(5), 2002.

[4] C. Roland and J.Palicot. Sélection de réseaux de radiocommunication accessibles pour un terminal récepteur multimode, 2002. Brevet num FR 99 15927.

[5] E2R projet. www.e2r2.motolabs.com.

[6] J. Mitola III. Software radio architecture : a mathematical perspective. *Communications, IEEE Journal on*, 17(4) :514 538, 1999.

[7] E. Buracchini. The software radio concept. *IEEE Communications Magazine*, 38(9) :138 143, 2000.

[8] FCC 036-222. Notice of rulemaking on cognitive radio, 2003.

[9] SDR Forum. www.sdrforum.org/.

[10] E.H. Armstrong. The super-heterodyne-its origin, development, and some recent improvements. In *Proceedings of the IRE*, volume 12(5), pages 539 – 552, 1924.

[11] P. Loumeau, J.F. Naviner, H. Petit, L. Naviner, and P. Desgreys. Analog to digital conversion : technical aspects. In *Annales des télécommunications*, volume 57, pages 338 – 385, 2002.

[12] D. Greifendorf, J. Stammen, S. Sappok, M. Ackeren van, and P. Jung. A novel hardware design paradigm for mobile "software defined radio" terminals. In *IEEE Seventh International Symposium on Spread Spectrum Techniques and Applications*, pages 828 – 831, Prague, Czech Republic, 2002.

[13] C. Moy, A. Nafkha, P. Leray, J. Delorme, J. Palicot, D. Nussbaumand K. Kalfallah, H. Callewaert, J. Martin, F. Clermidy, B. Mercier, R. Pacalet, D. Cibaud, P. Mary, and M. Ghozzi. Idromel : An open platform adressing advanced sdr challenges. *SDR Forum 2008*, 2008.

[14] X. Revés Ballesté, V. Marojevic, and A. Gelonch. Run-time and development framework for heterogeneous hardware radios. *Rapport*, September 2004.

[15] V. Marojevic, X. Revés Ballesté, and A. Gelonch. A computing resource management framework for software-defined radios. *Computers, IEEE Transactions on*, 57(10) :1399 – 1412, September 2008.

[16] David L. Tennenhouse and Vanu G. Bose. The spectrumware approach to wireless signal processing. *Wirel. Netw.*, 2(1) :1–12, 1996.

[17] Communication Research of Canada SCARI-Open. lien direct vers SCARI-Open depuis www.crc.ca/sdr.

[18] Zeligsoft. http ://www.zeligsoft.com/sdr-development-suite.cfm.

[19] PrismTech. http ://www.prismtechnologies.com/.

[20] Jeffrey H. Reed. *Software radio : a modern approach to radio engineering*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2002.

[21] Youping Zhao, Shiwen Mao, Jody Neel, and Jeffrey H. Reed. Performance evaluation of cognitive radios : metrics, utility functions, and methodologies, to appear Fall 2008.

[22] Ryuji Kohno. Special section on software defined radio technology and its applications, 2005.

[23] H. Harada. Software defined radio prototype for w-cdma and ieee802.11a wireless lan. *VTC2004-Fall Proc. of IEEE*, 6 :3919–3924, 2004.

[24] M. Beach, D. Bourse, M. Dillinger, R. Falk, T. Farnham, R. Navarro-prieto, and T. Wiebke. The european project trust : reconfigurable terminals and supporting networks. *Annales des télécommunications*, 57(7-8) :653–676, 2002.

[25] J. Thibault, E. Nicollet, and C. Demeure. Software defined radio architecture for cellular networks base stations : the sunbeam project. *Annales des télécommunications*, 57(7-8) :626–652, 2002.

[26] Jan M. Rabaey. *Low Power Design Methodologies*. The Springer International Series in Engineering and Computer Science, 1995.

[27] H. Wang, J.P. Delahaye, P. Leray, and J. Palicot. Managing dynamic reconfiguration on mimo decoder. *ipdps*, 0 :197, 2007.

[28] Xilinx Inc. Early access partial reconfiguration user guide, March 2006.

[29] J. D. Hadley. Design methodologies for partially reconfigured systems. In *FCCM '95 : Proceedings of the IEEE Symposium on FPGA's for Custom Computing Machines*, page 78, Washington, DC, USA, 1995. IEEE Computer Society.

[30] J. P. Delahaye. *Plate-forme hétérogène reconfigurable : application à la radio logicielle*. PhD thesis, Ecole SUPerieur d'Electricité, 2007.

[31] Xilinx. Microblaze processor reference guide (v 8.0), 2007.

[32] Altera. Nios 2 processor reference handbook (v 8.0), 2008.

[33] Gaisler. http ://www.gaisler.com.

[34] RJ. Lackey and DW. Upmal. Speakeasy : the military software radio. *IEEE Communications Magazine*, 33(5) :56–61, 1995.

[35] D.L. Lackey and V.G. Bose. The spectrumware approach to wireless signal processing. *Wireless Networks*, 2(1), 1996.

[36] JTRS. http ://sca.jpeojtrs.mil/.

[37] A. Pope. The corba reference guide : Understanding the common object request broker architecture, 1998.

[38] D. Lewine. Posix programmers guide, 1991.

[39] R. Marvie and P. Merle. Vers un modèle de composants pour cesure - le corba component model, November 2000.

[40] L. Pucker and J. Holt. Extending the sca core framework inside the modem architecture of a software defined radio, March 2004.

[41] J. Palicot and C. Roland. La radio logicielle : enjeux, contraintes et perspectives. In *REE. Revue de l'électricité et de l'électronique*, number 11, pages 60–67, Paris, FRANCE, 2001. Société de l'Electricité, de l'Electronique et des Technologies de l'Information et de la Communication (SEE).

[42] J. Palicot and C. Roland. Fft : a basic function for a reconfigurable receiver. In *10th International Conference on Telecommunications*, pages 898–902, Tahiti, 2003. Papeete.

[43] A. Royal and Peter Y. K. Cheung. Globally asynchronous locally synchronous fpga architectures. In *13th International Conference, FPL 2003*. Springer, 2003.

[44] SUPELEC, UPC, and RWTH. Report on hw/sw architecture of the flexible digital platform.

[45] E. Shiflet and L. Hansen. Fpga partial reconfiguration mitigates variability. *EE Times Online article*, 2006.

[46] L. Godard A. Nafkha C. Moy J.P. Delahaye, P. Leray. Designing a reconfigurable processing datapath for sdr over heterogeneous reconfigurable platforms. *SDR Forum Technical Conference*, 2007.

[47] C. Berrou. The ten-year-old turbo codes are entering into service. *Communications Magazine, IEEE*, 2003.

[48] S.J. Keyser, G.A. Miller, and E. Walker. An unpublished report submitted to the alfred p. sloan foundation. In *Cognitive Science*, New York, NY, USA, 1978.

[49] J. Hicks, A. MacKenzie, J. Neel, and J. Reed. A game theory perspective on interference avoidance. *Globecom 2004*, 2004.

[50] V. Srivastava, J. Neel, A. MacKenzie, J. Hicks, L.A. DaSilva, J.H. Reed, and R. Gilles. Using game theory to analyze wireless ad hoc networks. *IEEE Communications Surveys and Tutorials*, 7(4) :46 – 56, 2005.

[51] S. Haykin. Cognitive radio : brain-empowered wireless communications. *IEEE Journal on Selected Areas in Communications*, 23(2) :201 – 220, 2005.

[52] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks : A survey. *Computer Networks*, 50 :2127 – 2159, september 2006.

[53] S. Nandagopalan, C. Cordeiro, and K. Challapalli. Spectrum agile radios : Utilization and sensing architectures. *Proc.IEEE Dynamic Spectrum Access Networks (Dyspan)*, november 2005.

[54] G. Ganesan and G. Li. Cooperative spectrum sensing in cognitive radio networks (dyspan). *Proc. IEEE Dynamic Spectrum Access Networks*, november 2005.

[55] W. Horne, P. Weed, and D. Schaefer. Adaptive spectrum radio : A feasibility platform on the path to dynamic spectrum access. *Fifth Annual International Symposium on Advanced Radio Technologies*, march 2003.

[56] et al. C. Cordeiro. Ieee 802.22 : The first worldwide wireless standard based on cognitive radios. *Proc.IEEE Dynamic Spectrum Access Networks (Dyspan)*, november 2005.

[57] Nick R. Jennings. Coordination techniques for distributed artificial intelligence. *Foundations of distributed artificial intelligence*, pages 187–210, 1996.

[58] K.E. Nolan and L.E. Doyle. Principles of cognitive network teamwork. *Proceedings of the 51st SDR Forum General Meeting and 2006 Technical Conference*, november 2006.

[59] Chunyi Peng, Haitao Zheng, and Ben Y. Zhao. Utilization and fairness in spectrum assignment for opportunistic spectrum access. *Mob. Netw. Appl.*, 11(4) :555–576, 2006.

[60] Hadi Otrok, Noman Mohammed, Lingyu Wang, Mourad Debbabi, and Prabir Bhattacharya. A game-theoretic intrusion detection model for mobile ad hoc networks. *Comput. Commun.*, 31(4) :708–721, 2008.

[61] Pietro Michiardi and Refik Molva. A game theoretical approach to evaluate cooperation enforcement mechanisms in mobile ad hoc networks. In *In Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 3–5, 2003.

[62] A. Mackenzie. *Game Theory for Wireless Engineers (Synthesis Lectures on Communications)*. Morgan & Claypool Publishers, May 2006.

[63] E. Durfee, V. Lesser, and D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, KDE-1(1) :63–83, March 1989.

[64] B. Krenik. Clearing interference for cognitive radio. EETimes Online.

[65] J. Palicot, R.Hachemani, and C. Moy. La bulle sensorielle radio intelligente. *REE*, (09), 2007.

[66] Y. B. Reddy. Cognitive radio - genetic algorithm approach. In *ISIS International Symposium on Interdisciplinary Science*, volume 755 of *American Institute of Physics Conference Series*, pages 215–224, March 2005.

[67] C. J. Rieser. *Biologically inspired cognitive radio engine model utilizing distributed genetic algorithms for secure and robust wireless communications and networking*. PhD thesis, Virginia Polytechnic Institute and State University, 2004.

[68] C.J. Rieser, T.W. Rondeau, C.W. Bostian, and T.M. Gallagher. Cognitive radio testbed : further details and testing of a distributed genetic algorithm based cognitive engine for programmable radios. *Military Communications Conference, MILCOM 2004. IEEE*, 3 :1437 – 1443, 2004.

[69] Bruce A. Fette and Bruce Fette. *Cognitive Radio Technology (Communications Engineering)*, chapter 7. Newnes, 2006.

[70] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEEE*, 77(2), February 1989.

[71] Chang-Hyun Park, Sang-Won Kim, Sun-Min Lim, and Myung-Sun Song. Hmm based channel status predictor for cognitive radio. *Microwave Conference*, pages 1 – 4, December 2007.

[72] P. Jackson. *Introduction to Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[73] Carlos Andrés PEÑA REYES. *Coevolutionary fuzzy modeling*. PhD thesis, Lausanne, 2002.

[74] G. Mermoud, A. Upegui, C. Andrés Peña-Reyes, and E. Sanchez. A dynamically-reconfigurable fpga platform for evolving fuzzy systems. In Joan Cabestany, Alberto Prieto, and Francisco Sandoval Hernández, editors, *IWANN*, volume 3512 of *Lecture Notes in Computer Science*, pages 572–581. Springer, 2005.

[75] D. O. Hebb. *The Organization of Behavior : A Neuropsychological Theory*. Lawrence Erlbaum Associates, June 2002.

[76] Zhenyu Zhang and Xiaoyao Xie. Intelligent cognitive radio : Research on learning and evaluation of cr based on neural network. *Information and Communications Technology*, pages 33–37, 2007.

[77] C. Clancy, J. Hecker, E. Stuntebeck, and T. O'Shea. Applications of machine learning to cognitive radio networks. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 14(4) :47–52, 2007.

[78] Bruce A. Fette and Bruce Fette. *Cognitive Radio Technology (Communications Engineering)*, chapter 7. Newnes, 2006.

[79] C. J. Rieser. *Biologically Inspired Cognitive Radio Engine Model Utilizing Distributed Genetic Algorithms for Secure and Robust Wireless Communications and Networking*. PhD thesis, Virginia Polytechnic Institute and State University, 2004.

[80] E. Stuntebeck, T. OShea, J. Hecker, and T. Clancy. Architecture for an open-source cognitive radio. *SDR Forum Technical Conference (SDR)*, 2006.

[81] L. Godard, C. Moy, and J. Palicot. From a configuration management to a cognitive radio management of sdr systems. *CROWNCOM*, 2006.

[82] S. Leriche. Vers un modèle d'agent flexible. In *Journées Multi-Agent et Composant (JMAC 2006)*, Nîmes, France, mars 2006. École des Mines d'Alès.

[83] N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, (1) :275–306, 1998.

[84] G. Gielen and E. Goris. Reconfigurable front-end architectures and a/d converters for flexible wireless transceivers for 4g radios. *Emerging Technologies : Circuits and Systems for 4G Mobile Wireless Communications*, pages 13 – 18, 2005.

[85] Bruce E. Carey-Smith, Paul A. Warr, and Mark A. Beach. Mems-driven flexible filters for cognitive radio. *IST Mobile and Wireless Communications Summit*, 2005.

[86] G. Collin, T. Ditchi, and E. Geron. Optimisation d'un réseau de communication dect à l'aide d'une antenne adaptative, 2006.

[87] AMICOM. http ://www.amicom.info/.

[88] ACE. http ://www.ist-ace.org/.

[89] ARROW. http ://www.wtc-consult.com/english/home/index.html.

[90] C. Moy A. Kountouris. Reconfiguration in software radio systems. *2nd Karlsruhe Workshop on Software Radios*, 2002.

[91] MDA Guide Version 1.0.1. www.omg.org/docs/omg/03-06-01.pdf, 2003.

[92] MDA. http ://www.omg.org/mda/.

[93] OMG. http ://www.omg.org/technology/documents/formal/uml.htm.

[94] version 2.0 MetaObject Facility (MOF). http ://www.omg.org/mof/.

[95] version 2.1.1 XML Metadata Interchange (XMI). http ://www.omg.org/technology/documents/formal/xmi.htm.

[96] version 1.1 Common Warehouse Metamodel (CWM). http ://www.omg.org/technology/documents/formal/cwm.htm.

[97] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose. Eclipse modeling framework. *Addison Wesley Professional*, 2003.

[98] SOFTEAM. http ://www.softeam.fr/pdf/fr/uml_profiles.pdf.

[99] Sanford A. Friedenthal and Roger Burkhart. Extending uml from software to systems, 2003.

[100] OMG Systems Modeling Language. http ://www.omgsysml.org/.

[101] OMG. www.omgmarte.org/tutorial.htm.

[102] MOPCOM. www.mopcom.fr/.

[103] P.A. Muller, F. Fleurey, and J.M. Jézéquel. Weaving executability into object-oriented meta-languages. *Proceedings of MODELS/UML'2005*, 2005.

[104] Triskell team. www.kermeta.org/.

[105] Meta Object Facility (MOF). Core specification omg available specification. Version 2.0 formal/06-01-01.

[106] Klaus Fischer, Jorg P. Muller, and Markus Pischel. Unifying control in a layered agent architecture. In *In IJCAI95, Agent Theory, Architecture and Language Workshop 95*, 1995.

[107] Jörg P. Müller and Markus Pischel. The agent architecture inteRRaP : Concept and application. Technical report, German Research Center for Artificial Intelligence, 1993. RR 93-26.

[108] D. Elleouet. *Méthodes de modélisation, d'estimation et d'optimisation du TDSI pour la conception de systèmes reconfigurables de type FPGA*. PhD thesis, Université de Rennes 1, 2007.

[109] R. Hachemani, J. Palicot, and C. Moy. A new standard recognition sensor for cognitive radio termonals. In *European Signal and Image Processing Conference (EUSIPCO), Poznan, 03/09/2007-07/09/2007*. EURASIP, 2007.

[110] Shuji Goto and Tadashi Senoo. Micro fuel cell system for mobile consumer electronic devices. In *10th Annual International Conference SMALL FUEL CELLS - Portable and Micro Fuel Cells for Military and Commercial Applications*, Atanta GA, USA, May 2008.

[111] Franck Fleurey. *Langage et méthode pour une ingénierie des modèles fiable*. PhD thesis, Université de Rennes 1, 2006.

[112] P.-A Muller, F. Fleurey, D. Vojtisek, Z. Drey, D. Pollet, F. Fondement, P. Studer, and J.-M. Jézéquel. On executable meta-languages applied to model transformations. *Model Transformation In Practice (MTIP) workshop held in conjunction with MoDELS/UML 2005*, 2005.

Résumé

Ce travail porte sur la mise en œuvre d'une architecture de gestion pour équipement radio cognitif en vue d'applications dans le domaine des radiocommunications. Ce projet pluridisciplinaire regroupe des domaines de compétence variés tels que : l'électronique, l'informatique et les sciences cognitives. L'architecture retenue porte le nom HDCRAM (Hierarchical and Distributed Cognitive Radio Management). HDCRAM est distribuée de façon hiérarchique au sein de l'équipement sur trois niveaux d'abstraction. Cette distribution hiérarchique permet de prendre en compte l'une des problématiques du domaine qui est l'hétérogénéité des plateformes d'exécution cible. HDCRAM propose une gestion fine tant du point de vue des mécanismes de reconfiguration que de la gestion des prises de décision menant à une reconfiguration de tout ou partie du système. Le cadre applicatif de cette architecture étant un domaine où la part logicielle devient de plus en plus prédominante sur la part matérielle, il est nécessaire de définir une interface commune. Ceci afin de faciliter le portage des logiciels sur un parc d'équipement en constante augmentation et par nature hétérogène. Grâce à une modélisation à haut niveau d'abstraction par l'utilisation du langage de modélisation UML nous avons pu définir HDCRAM de façon totalement indépendante des contraintes matérielles ce qui offre une possibilité étendue en termes de réutilisabilité et de modularité. Le choix de doter cette modélisation d'un métalangage de programmation exécutable tel que Kermeta permet, en plus de la modélisation à haut niveau d'abstraction, une simulation fonctionnelle de HDCRAM via une description comportementale.

Abstract

This work focuses on the implementation of a management architecture for cognitive radio equipment for applications in the field of radiocommunications. This multidisciplinary project brings together diverse areas of expertise such as electronics, information technology and cognitive sciences. The architecture is named HDCRAM (Hierarchical and Distributed Cognitive Radio Management). HDCRAM is hierarchically distributed in the equipment on three levels of abstraction. This distribution can take into account one of the problems of that field of activity which is the heterogeneity of execution platforms. Thanks to a precise management for both reconfiguration and decision-making leading to a reconfiguration of all or part of the system. As the framework application of this architecture is an area where electronic components are controlled by an increasingly software part, it is necessary to define a common interface. This is done to facilitate the porting of software into a growing installed base. Through the use of UML language, for high level of abstraction modeling, we define a platform independent model of HDCRAM which offers an extended opportunity in terms of reusability and modularity. The choice to use an executable metamodeling language as Kermeta for HDCRAM allows describing both structural and behavioral part of our architecture and gives the opportunity to make functional simulation.